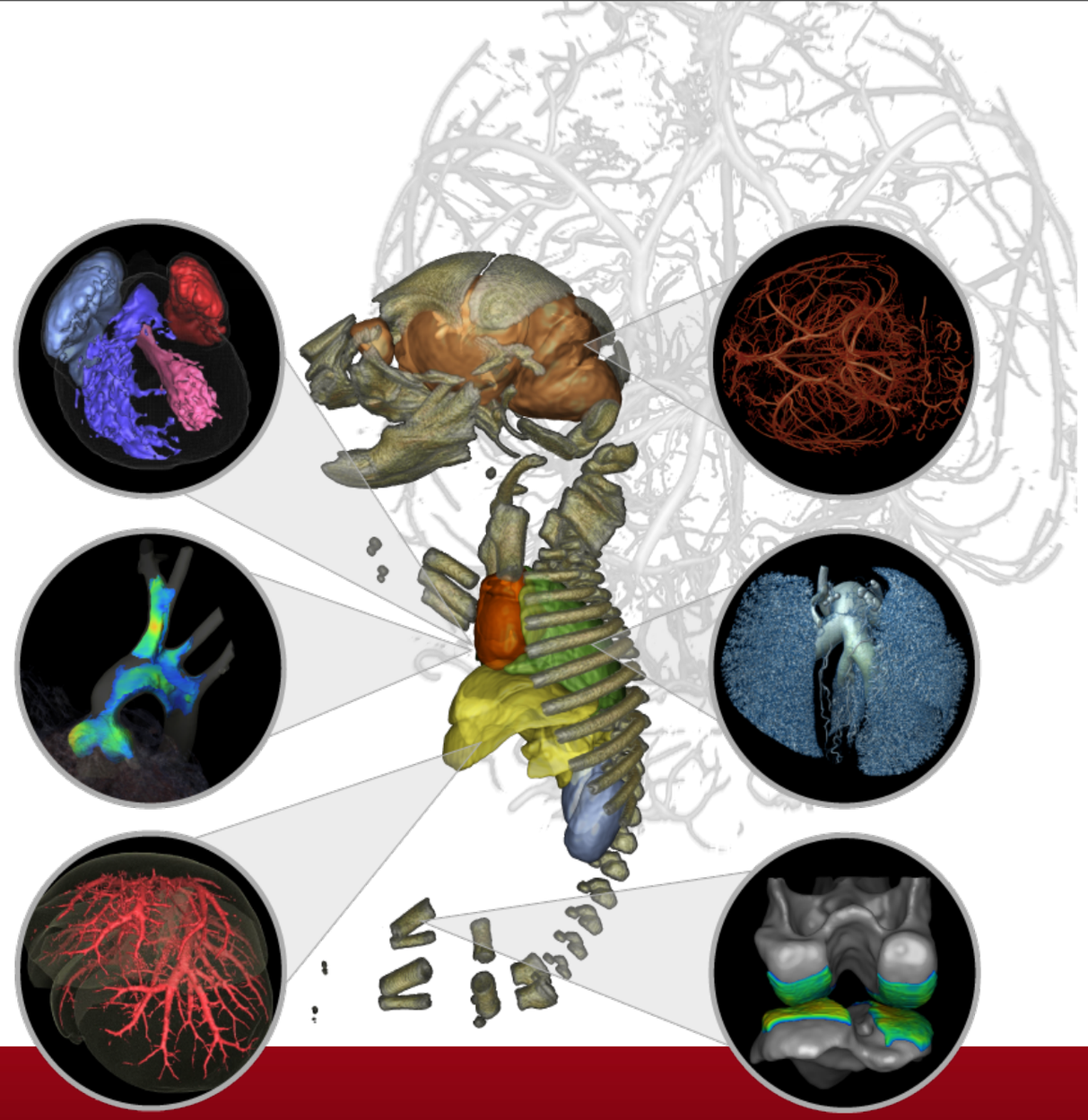
Seg3D Insights

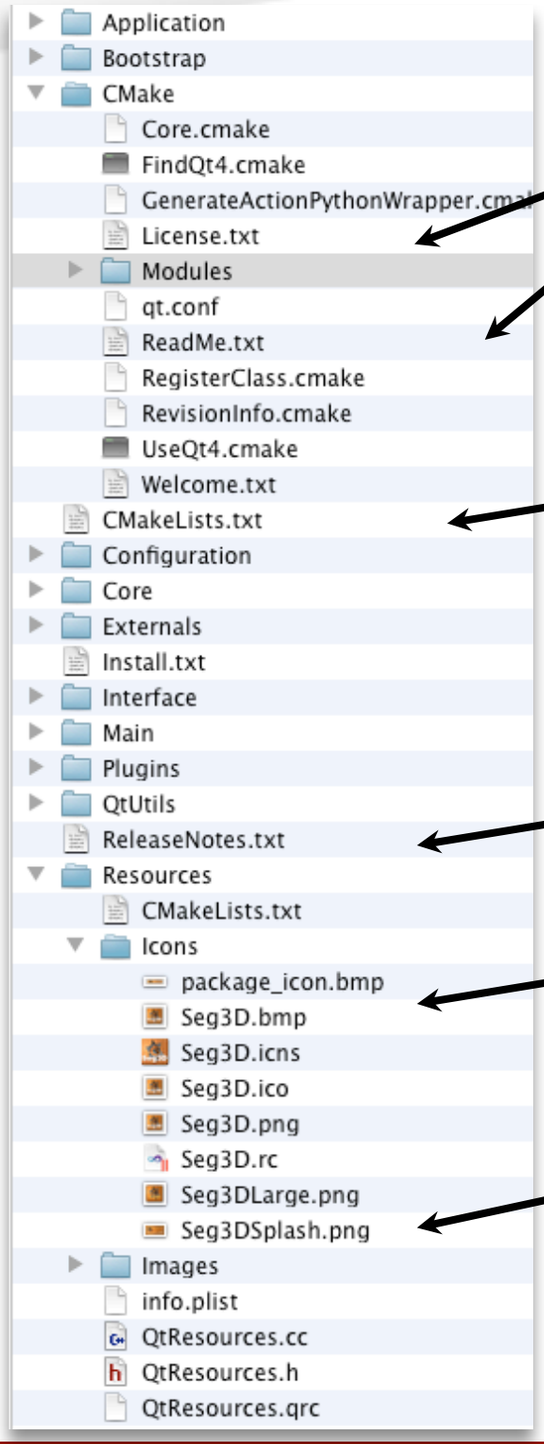
Architectural Need to Knows



Numira Biosciences

The New Vision of Discovery

Versioning/Packaging of Seg3D



Files with Welcome information and Program description (CPack)

Main configuration file that controls Application name and Application version:
We have used “Apple style” release numbers. All of them are version 2, and we use the minor number to indicate major updates and we use the patch number to count bug releases.

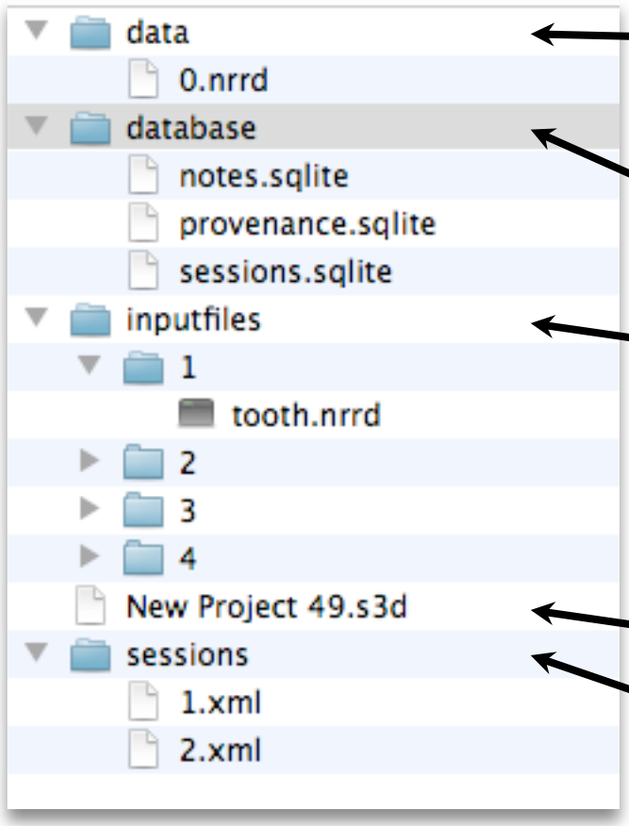
Release notes for installer

Icons for application: We use Icon Composer.app from the Mac to make them all.

Splash Screen

Architecture of Seg3D: Project Folders

Example of a project directory



Data files used in the sessions. Each volume is only stored once and masks are grouped together into one nrrd file

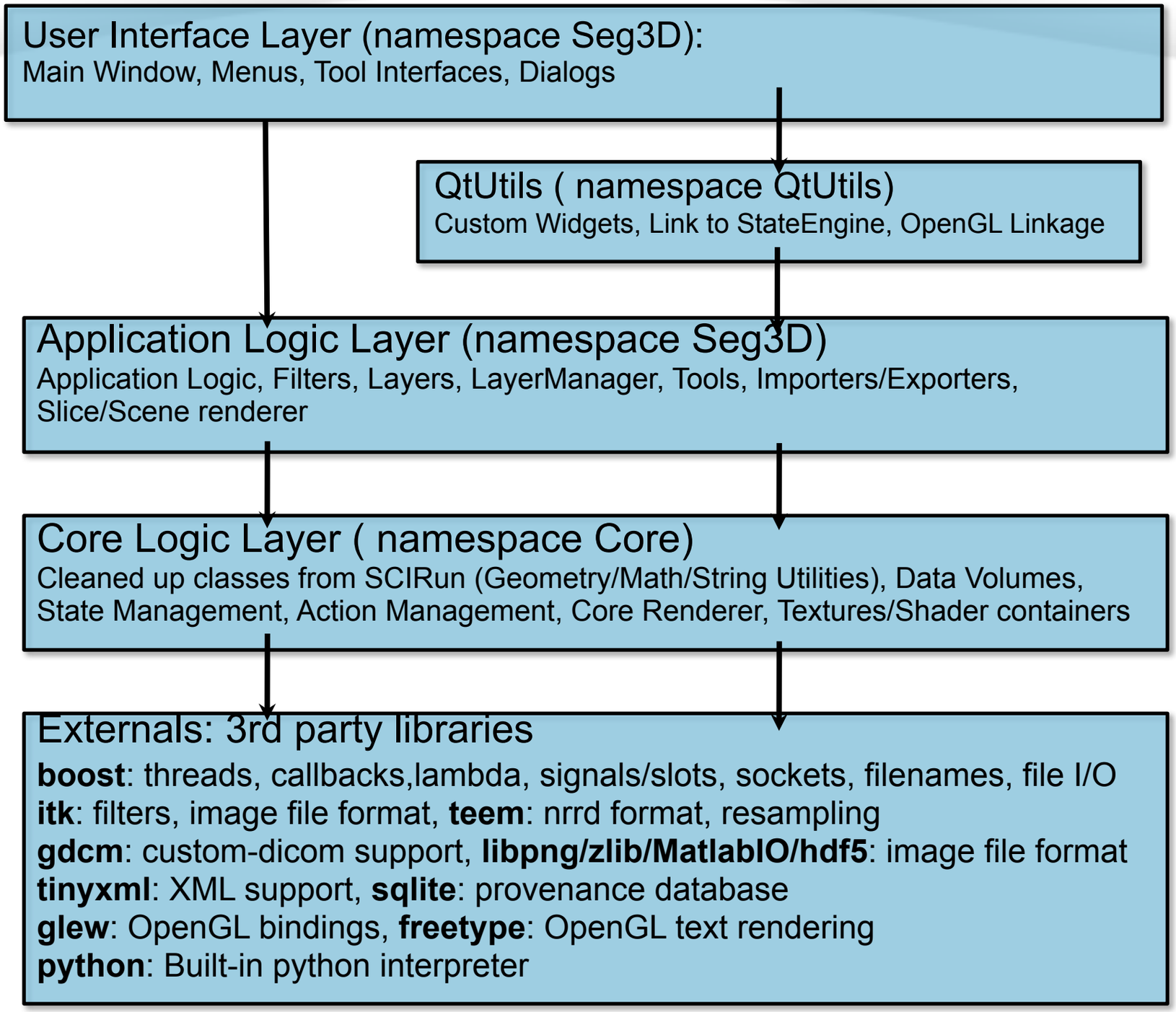
Database files

Provenance requires play back hence a copy is made of all the files that are imported, so all information is stored (GLP)

Main project file (XML format)

Each session is listed as its own file

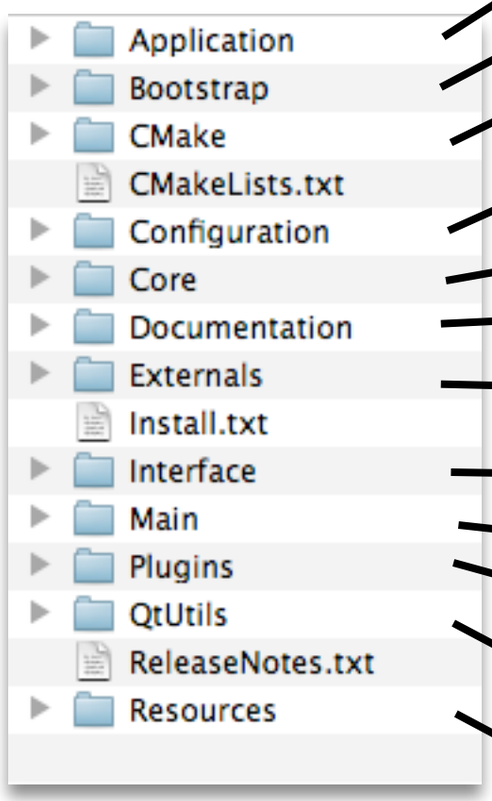
Architecture of Seg3D: Layered Approach



Lower layers do not depend on code in layers above:

Hence you could build a headless Seg3D version by taking the top layer off.

Architecture of Seg3D: Directory Structure



Application Layer

Standalone code for generating code on the fly, e.g. shaders

Extensions to the CMake system and config files for CPack

Templates for code that is automatically generated, e.g. plugins, version information

Core Layer

Documentation (slides)

Third party Libraries

Interface Layer

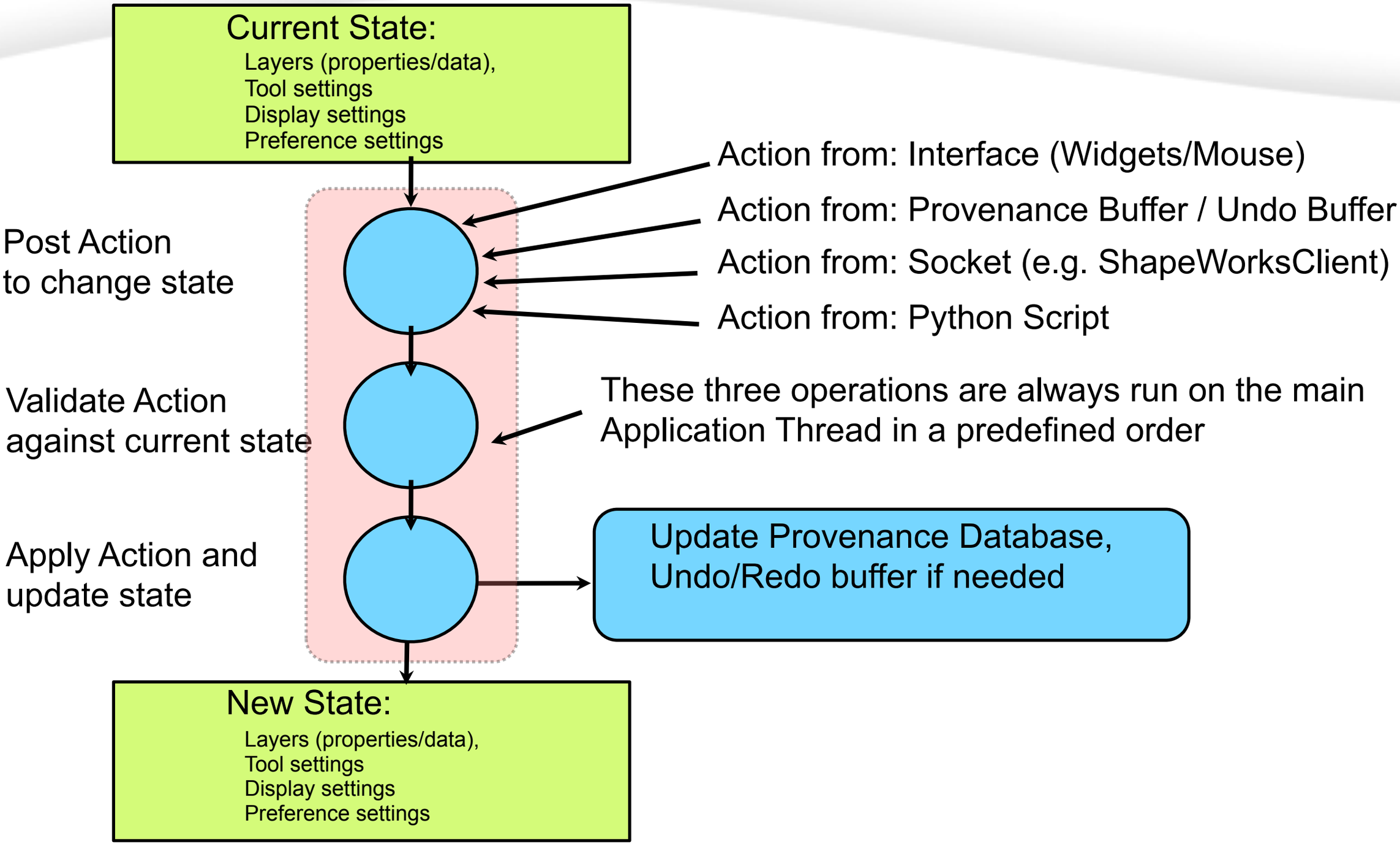
Main program that starts the interface layer

Plugins: Code in the sub directories of this one will be automatically included in the CMake system.

Custom Qt Widgets

Images/Icons used by the visual interface

Architecture of Seg3D: Action Mechanism



Architecture of Seg3D: State Mechanism 1

StateEngine (Singleton)

Pointers to where state is stored
Central mutex to lock all state

Code for saving/loading state
(Session Control)

There is one StateEngine in the system that controls all state

StateHandler (Base Class)

Hooks for pre/post loading/saving state
Signals for when state is changed
Pointers to individual state variables
Whether information is stored in sessions

Many classes in Seg3D derive from StateHandler

Seg3d::Layer

StateHandler (Base Class)

Hooks for pre/post loading/saving state
Signals for when state is changed
Pointers to individual state variables

StateVariable (Many different Classes)

Signals for when it is changed
Code of serializing data
Code for setting new values
Code for checking constraints (e.g. options/limits)

There are different state variables for different data types:

- StateRangedDouble
- StateOption
- StateString
- StateView3D
- ...

Architecture of Seg3D: State Mechanism 2

Session files are generated using tinyXML.

Each StateHandler has an unique id, if it is a singleton class it will have name like **toolmanager**; if it is an instance of a class it will have name like **layer_1**.

Inside each StateHandler all states have an unique id.

Each state is stored like
<State id=”name of state”> value </State>

Object values are serialized using two functions:

bool ImportFromString(std::string text, T& value);
std::string ExportToString(T& value);

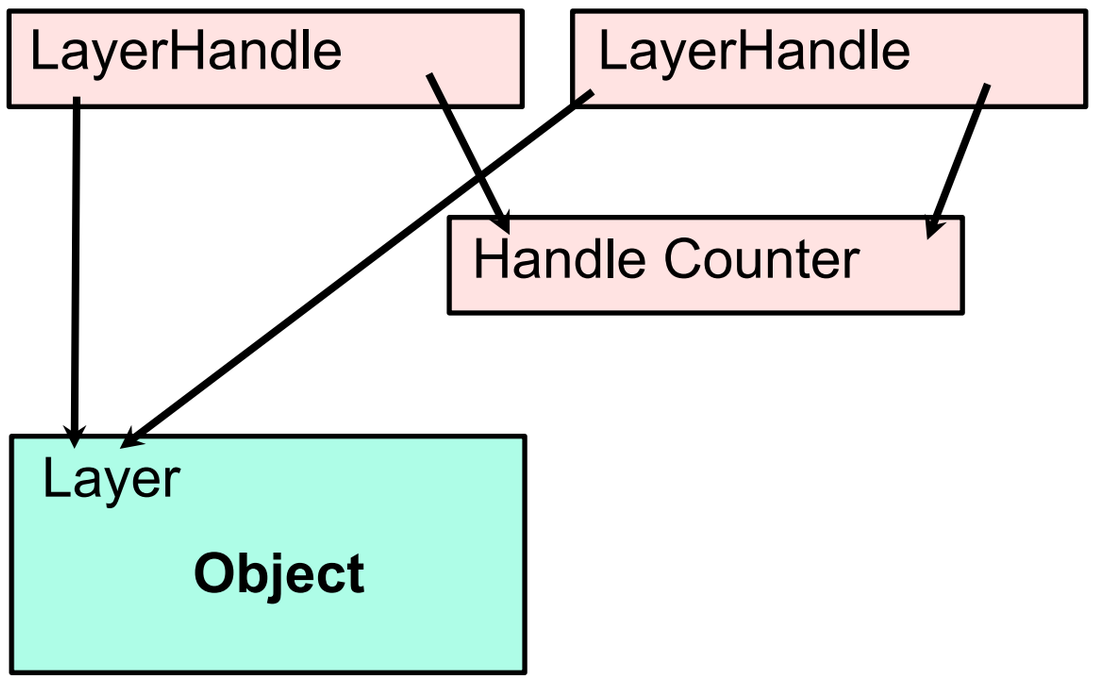
For all basic types these functions have been defined in Core/Utils/StringUtil.h

Example session file

```
<?xml version="1.0" ?>
<Seg3D2 major_version="2" minor_version="0" patch_version="3">
    <toolmanager version="1">
        <State id="active_tool">thresholdtool_0</State>
        <State id="disable_tools">false</State>
        <tools />
    </toolmanager>
    <view version="1">
        <State id="active_axial_viewer">3</State>
        <State id="active_coronal_viewer">0</State>
        <State id="active_sagittal_viewer">-1</State>
        <State id="active_viewer">4</State>
        <State id="cp1_distance">0</State>
        <State id="cp1_enable">false</State>
        <State id="cp1_reverse_norm">true</State>
        <State id="cp1_x">1</State>
        <State id="cp1_y">0</State>
        <State id="cp1_z">0</State>
        <State id="cp2_distance">0</State>
        <State id="cp2_enable">false</State>
        <State id="cp2_reverse_norm">true</State>
        <State id="cp2_x">0</State>
        <State id="cp2_y">1</State>
        <State id="cp2_z">0</State>
        <State id="cp3_distance">0</State>
        <State id="cp3_enable">false</State>
        <State id="cp3_reverse_norm">true</State>
        <State id="cp3_x">0</State>
        <State id="cp3_y">0</State>
        <State id="cp3_z">1</State>
```

Architecture of Seg3D: Memory Management

Handle mechanism:



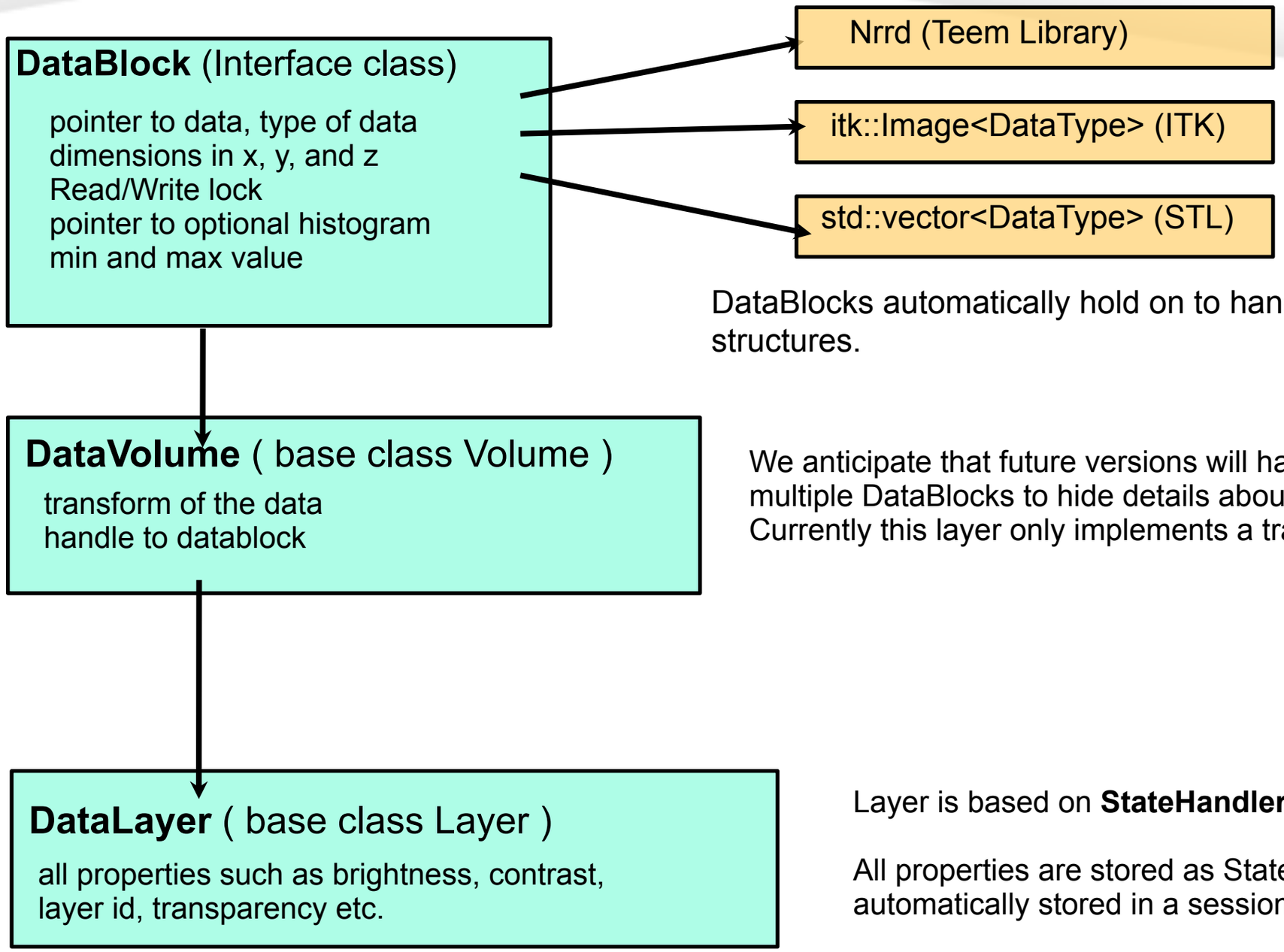
Memory management based on following assumptions:

- Program will be deployed on a **64bit** architecture where memory fragmentation is no big issue.
- All major objects are reference counted using boost's thread safe model.
- The last object to hold on to a handle will automatically delete the object when it deletes the handle.

Implementation details:

```
// Forward Define Layer class
class Layer;

// Define a handle for memory management
typedef boost::shared_ptr<Layer> LayerHandle;
```

Architecture of Seg3D: Data Layer structures



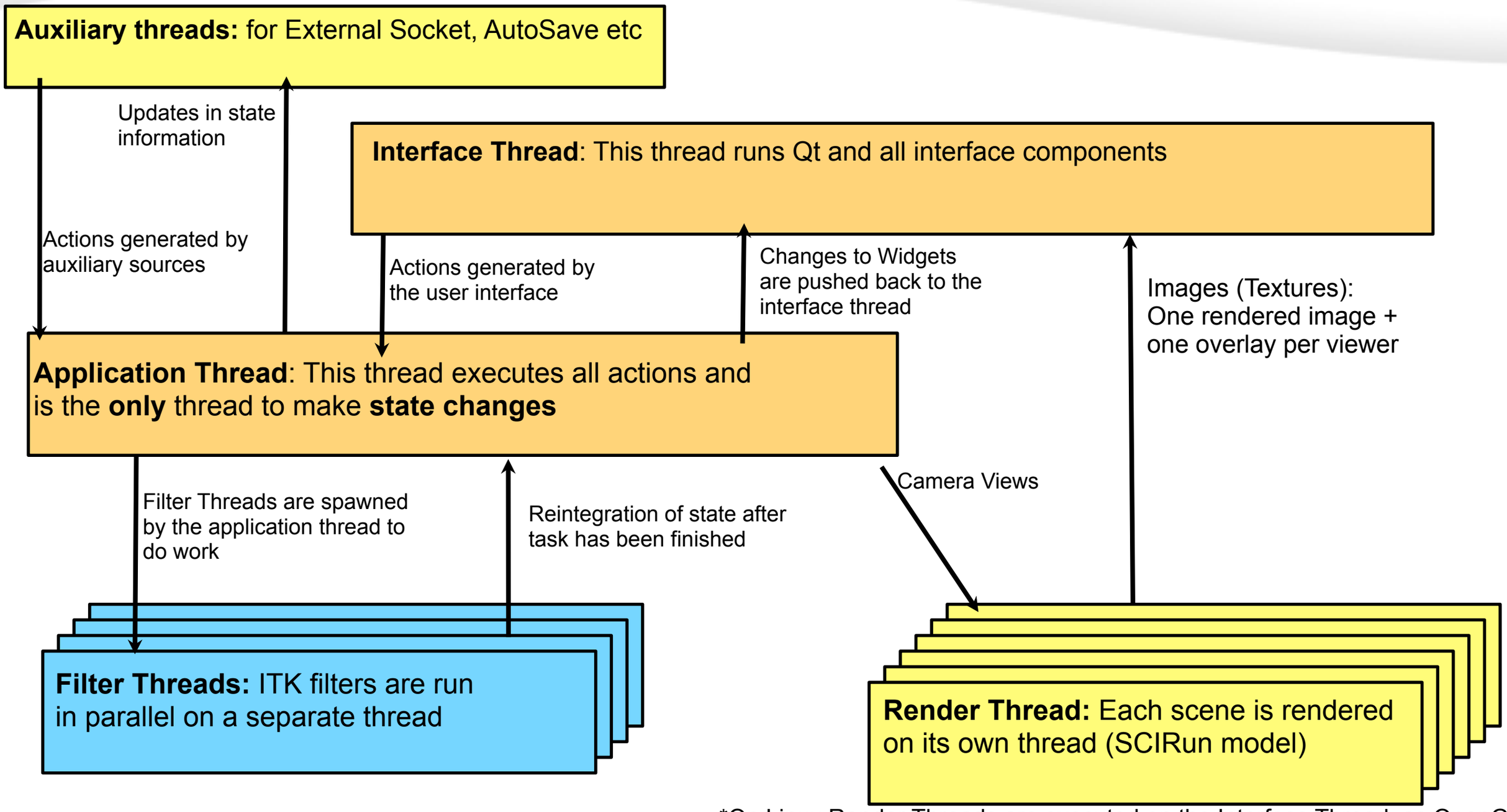
DataBlocks automatically hold on to handles of the underlying data structures.

We anticipate that future versions will have Volumes that point to multiple DataBlocks to hide details about bricking, slabbing etc. Currently this layer only implements a transform.

Layer is based on **StateHandler**

All properties are stored as StateVariables so they are automatically stored in a session.

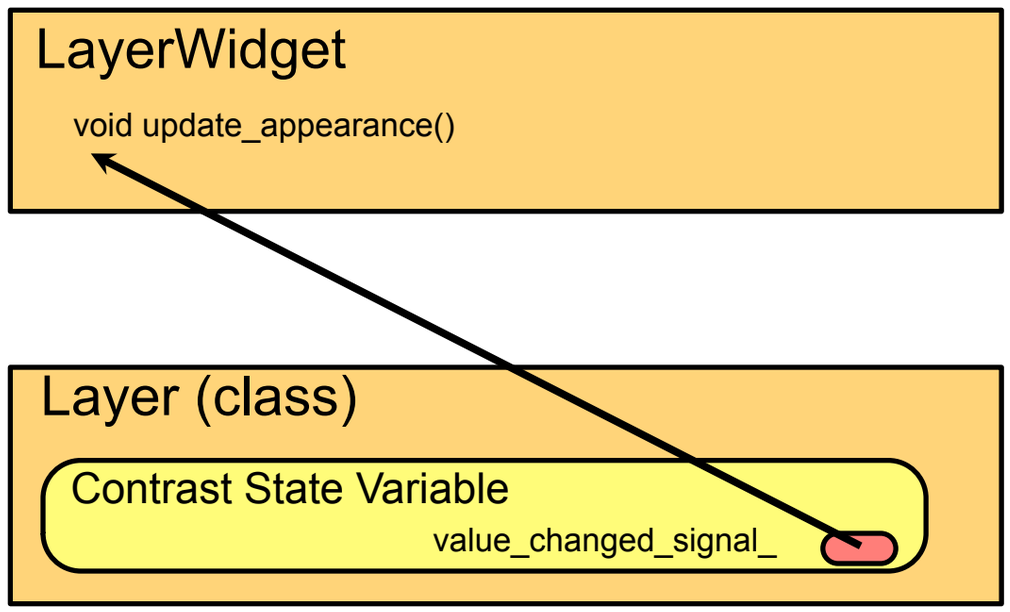
Architecture of Seg3D: Threading model



*On Linux Render Threads are executed on the Interface Thread, as OpenGL drivers were found not to be thread-safe in most cases.

Architecture of Seg3D: Signal/Slots

Objects signal when they have an event



Components on the top level of the architecture can listen to events of underlying components without the underlying components needing to known about the logic above.

With signals and slots the core classes do not have to know about the interface, they just need to signal events.

Signals and slots in boost

Defining a signal:

```
class MyClass
{
public:
typedef boost::signals2< void ( void ) > value_changed_signal_type;
value_changed_signal_type value_changed_signal_;
}
```

Triggering a signal:

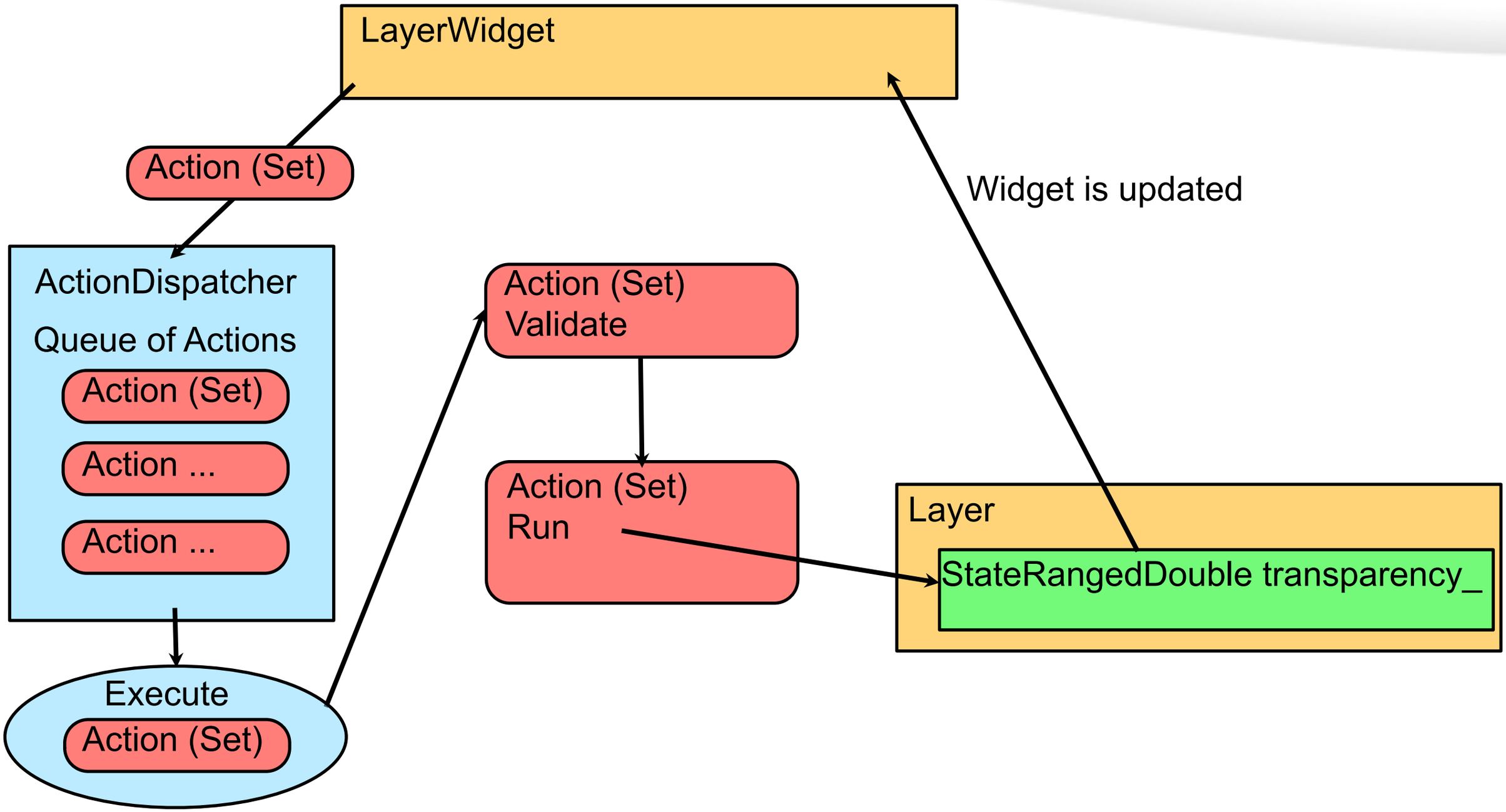
```
MyClassHandle my_class( new MyClass );
my_class-> value_changed_signal_();
```

Binding a slot to a signal:

```
my_class-> value_changed_signal_.connect(
boost::bind( &LayerWidget::update_appearance, my_layer  ) );
```

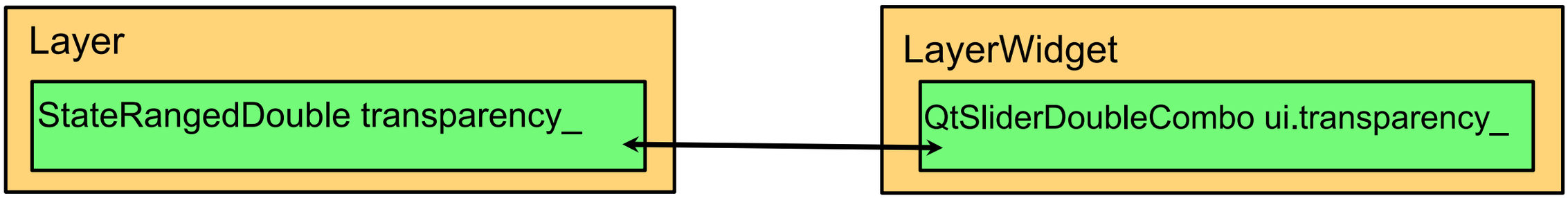
Architecture of Seg3D: Actions

Transparency Slider is moved



Architecture of Seg3D: QtUtils

Most logic of posting actions/validation and setting up automatic updates to widgets is handled by QtUtils::QtBridge class



QtUtils::QtBridge::Connect(widget, state variable)

Also automatic hiding and enabling of widgets

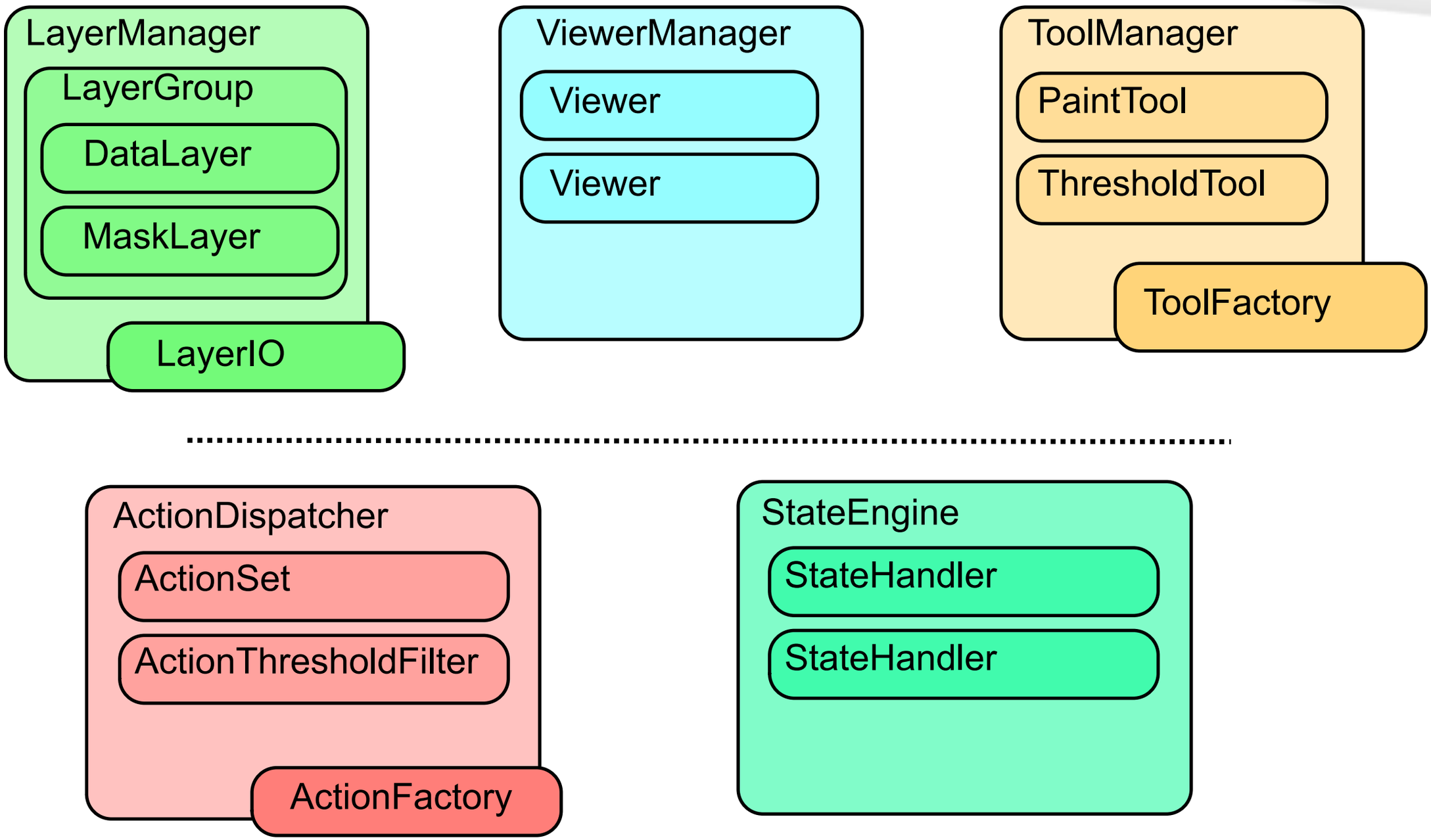
QtUtils::QtBridge::Enable(widget, boolean state variable)
QtUtils::QtBridge::Enable(widget, state variables, lambda expression)

QtUtils::QtBridge::Show(widget, boolean state variable)

All logic that most filters show in the interface is automatically generated using the QtBridge, by specifying simple rules or complex rules using lambda expressions.

Architecture of Seg3D: Class Hierarchy

The key to most components is a singleton manager class that provides access to its components



Architecture of Seg3D: Plugin Architecture

Plugin architecture support for:

LayerImporters
Tools
ToolInterfaces
Actions

All Plugins are **Compile-Time** plugins, hence a recompile is needed to add plugins.

As Seg3D is mostly statically linked, a special system is available in the CMake system to ensure that symbols are added to the executable and plugins are registered with the right factories.

Special Instructions in CMake

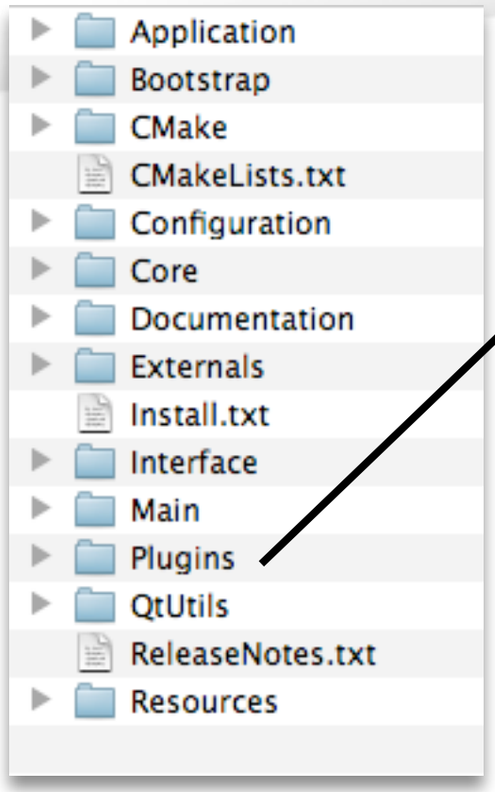
```
# Register action classes
REGISTER_LIBRARY_AND_CLASSES(Application_Filters
  ${APPLICATION_FILTERS_ACTIONS_SRCS})
```

CMake automatically creates function to register all components

ClassRegistration.h

```
extern void register_ActionAdd();
extern void register_ActionClear();
extern void register_ActionGet();

...


void RegisterClasses()
{
   register_ActionAdd();
   register_ActionClear();
   register_ActionGet();
  ....
}
```

Architecture of Seg3D: Plugin directory



Each sub directory in here is an extension package

src/Plugins/MyPackage/

The only required file is a **CMakeList.txt** in this directory.

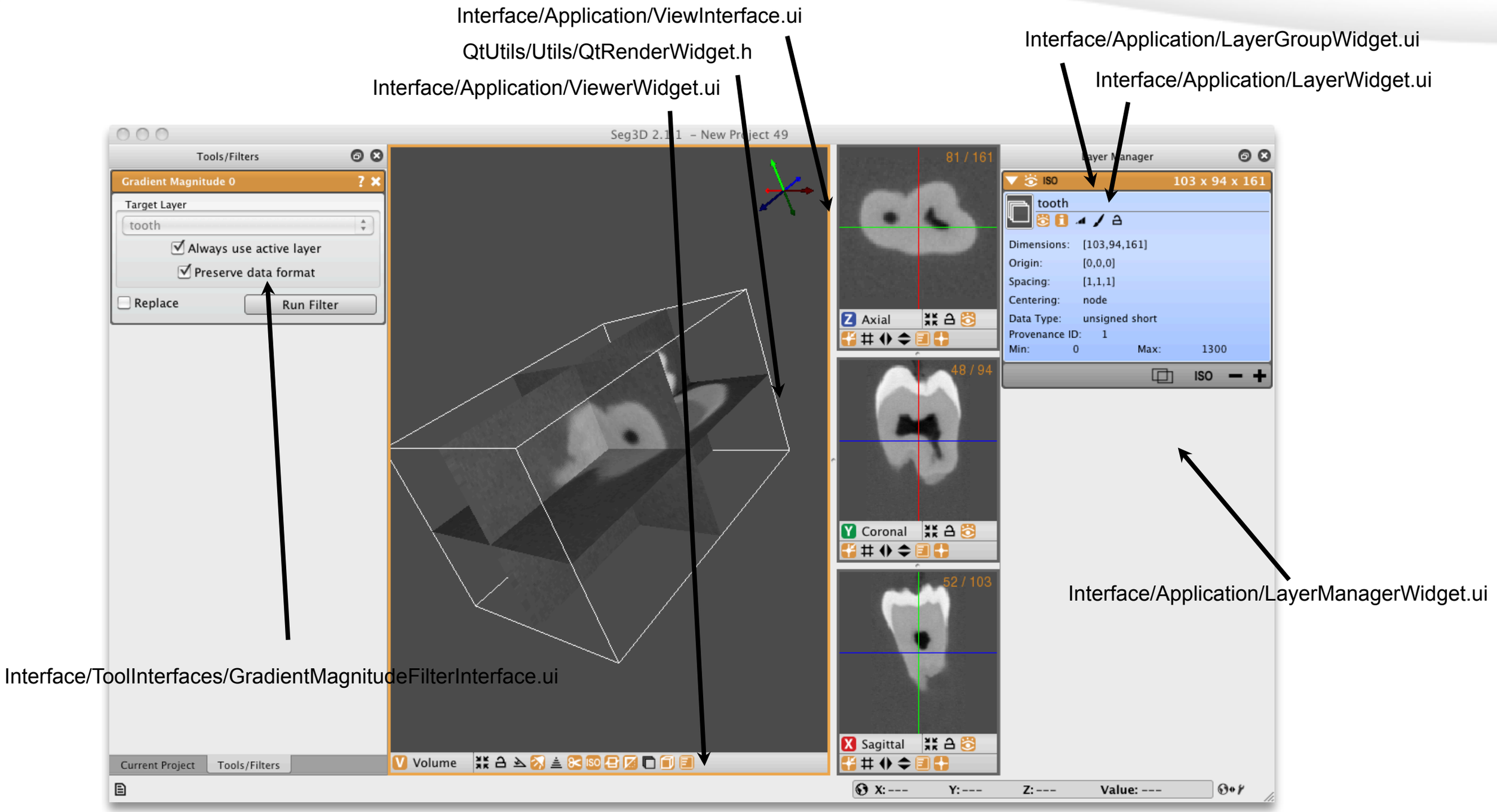
The Plugins are included in CMake just before Main is configured, hence one has access to all other components.

The layout of a package can be arbitrary.

* New Tools can be configured inside a package, by deriving a class from the Tool class and adding the registration macro in the CMake file and the .cc file.
* New Actions/Importers can be registered in a plugin.
* A plugin can alter CMake settings such as application name and version.
* A plugin can register a function to initialize new components and add extra observers.

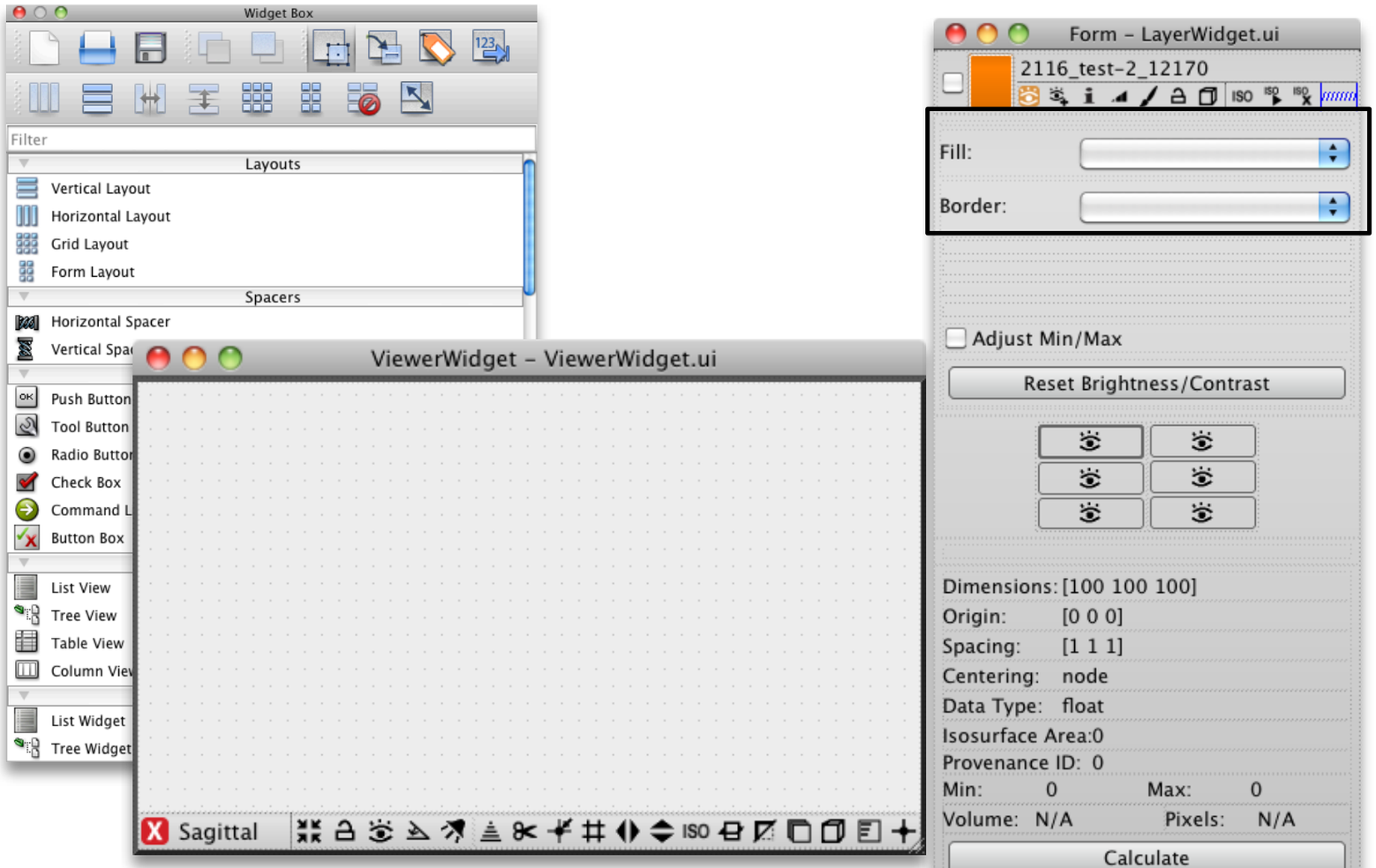
Architecture of Seg3D: Qt Interface 1

The interface is split into many components that are reusable



Architecture of Seg3D: Qt Interface 2

All major components of the UI were designed with QtDesigner



Option that are not needed are programmatically switched off using the tags added to the buttons

Frames are later programmatically hidden by collapsing the widgets that group options together.

Qt Need to Know Issues:

Font sizes on Mac and Windows do not match, leave at default value or programmatically change them later.

Default layout spacing is different on Mac and Windows, hence we always specify the layout spacings.

For Option boxes we often set spacing to Ignored and for all others we set it to Preferred to ensure proper scaling.

All of our UI files have one StyleSheet in widget that frames all widgets together.

UIs often require experimenting with layout settings until it looks good on all platforms.

Architecture of Seg3D: Qt Interface 3

Almost all Qt Designer widgets have been incorporated using a private class concept.

Header file

```
....

namespace Seg3D
{

class GradientMagnitudeFilterInterfacePrivate;

class GradientMagnitudeFilterInterface : public ToolWidget
{
Q_OBJECT

// -- Constructor/destructor --
public:
    GradientMagnitudeFilterInterface();
    virtual ~GradientMagnitudeFilterInterface();

// -- create interface --
public:
    // BUILD_WIDGET:
    // This function builds the actual GUI
    virtual bool build_widget( QFrame* frame );

// -- filter internals --
private:
    boost::shared_ptr< GradientMagnitudeFilterInterfacePrivate
> private_;

};

}
```

Implementation file

```
// QtGui includes
#include "ui_GradientMagnitudeFilterInterface.h"

// Interface includes
#include <Interface/ToolInterface/GradientMagnitudeFilterInterface.h>

SCI_REGISTER_TOOLINTERFACE( Seg3D, GradientMagnitudeFilterInterface )

namespace Seg3D
{

class GradientMagnitudeFilterInterfacePrivate
{
public:
    Ui::GradientMagnitudeFilterInterface ui_;
};

GradientMagnitudeFilterInterface::GradientMagnitudeFilterInterface() :
    private_( new GradientMagnitudeFilterInterfacePrivate )
{
}

// build the interface and connect it to the state manager
bool GradientMagnitudeFilterInterface::build_widget( QFrame* frame )
{
    //Step 1 - build the Qt GUI Widget
    this->private_->ui_.setupUi( frame );

.....
}

}
```

Architecture of Seg3D: Anatomy of a Filter

Interface files (base class ToolInterface)

Interface/ToolInterface/GradientMagnitudeFilterInterface.cc
Interface/ToolInterface/GradientMagnitudeFilterInterface.h
Interface/ToolInterface/GradientMagnitudeFilterInterface.ui

These files wrap an interface build in QtDesigner. The ui file is directly produced by QtDesigner. The interface cc file connects the widgets to the underlying state variables and spells out the interface logic.

Tool files (base class Tool)

Application/Tools/GradientMagnitudeFilter.cc
Application/Tools/GradientMagnitudeFilter.h

The Tool files declare the state underlying the filter, they declare the parameters used and link the state to the current session.

Action files (base class LayerAction)

Application/Filters/Actions/ActionGradientMagnitudeFilter.cc
Application/Filters/Actions/ActionGradientMagnitudeFilter.h

The action that does the actual filtering. An action records the current parameters and locks the layers and performs the filtering.

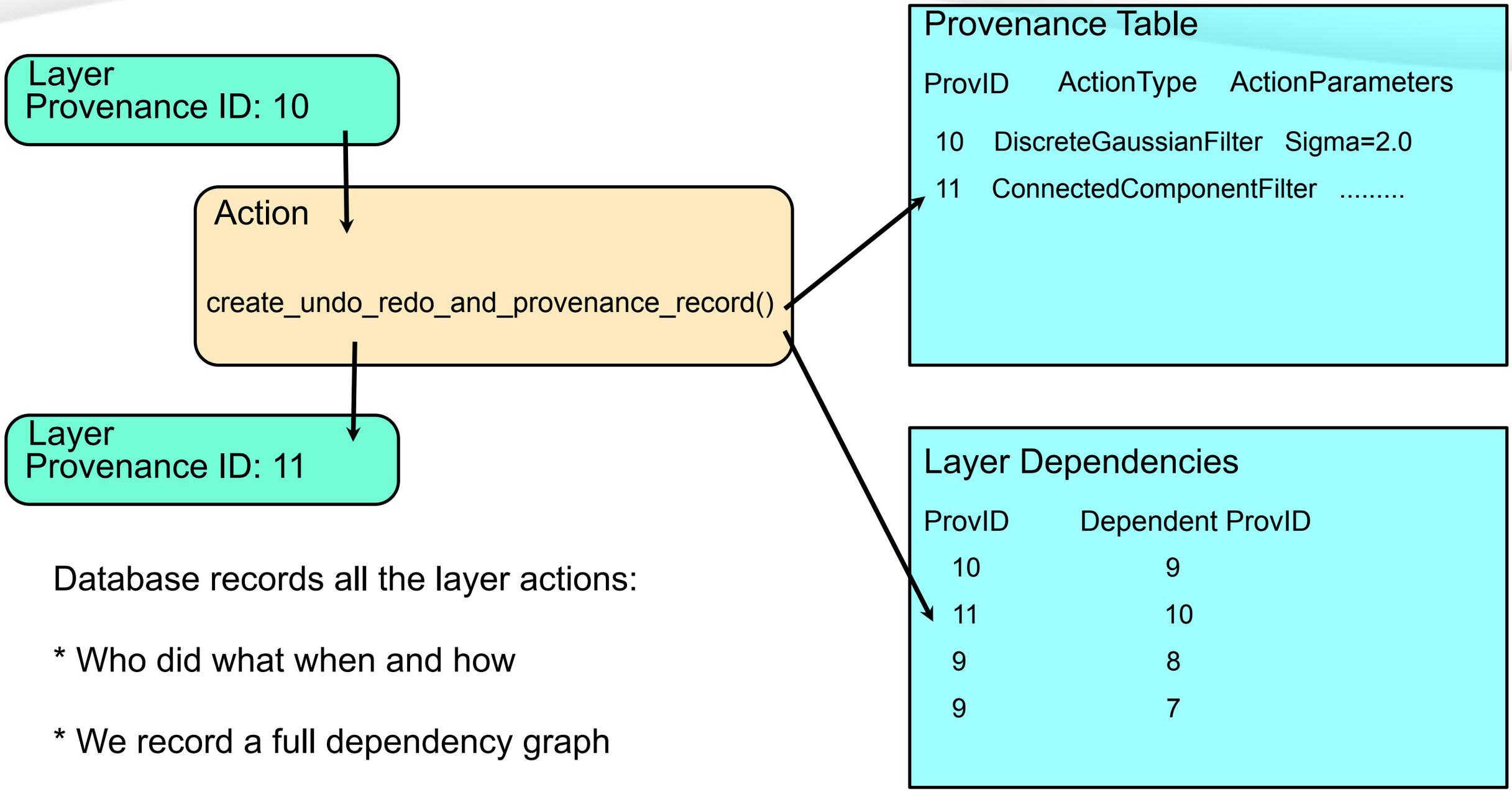
Architecture of Seg3D: Python

- Currently used inside the provenance playback mechanism
- Setup to do scripting in the future
- Currently not exposed through the interface
- Only way to access python currently is through the python console

All actions have been made into python commands that can be run on the command line.

1. Import a layer
   importlayer(filename='E:/Datasets/tooth.nrrd', importer='nrrd')
2. Run discrete Gaussian filter
   discretegaussianfilter(layerid='layer_1', replace=False)
3. Undo an action
   undo()
4. Redo an action
   redo()

Architecture of Seg3D: Provenance Engine



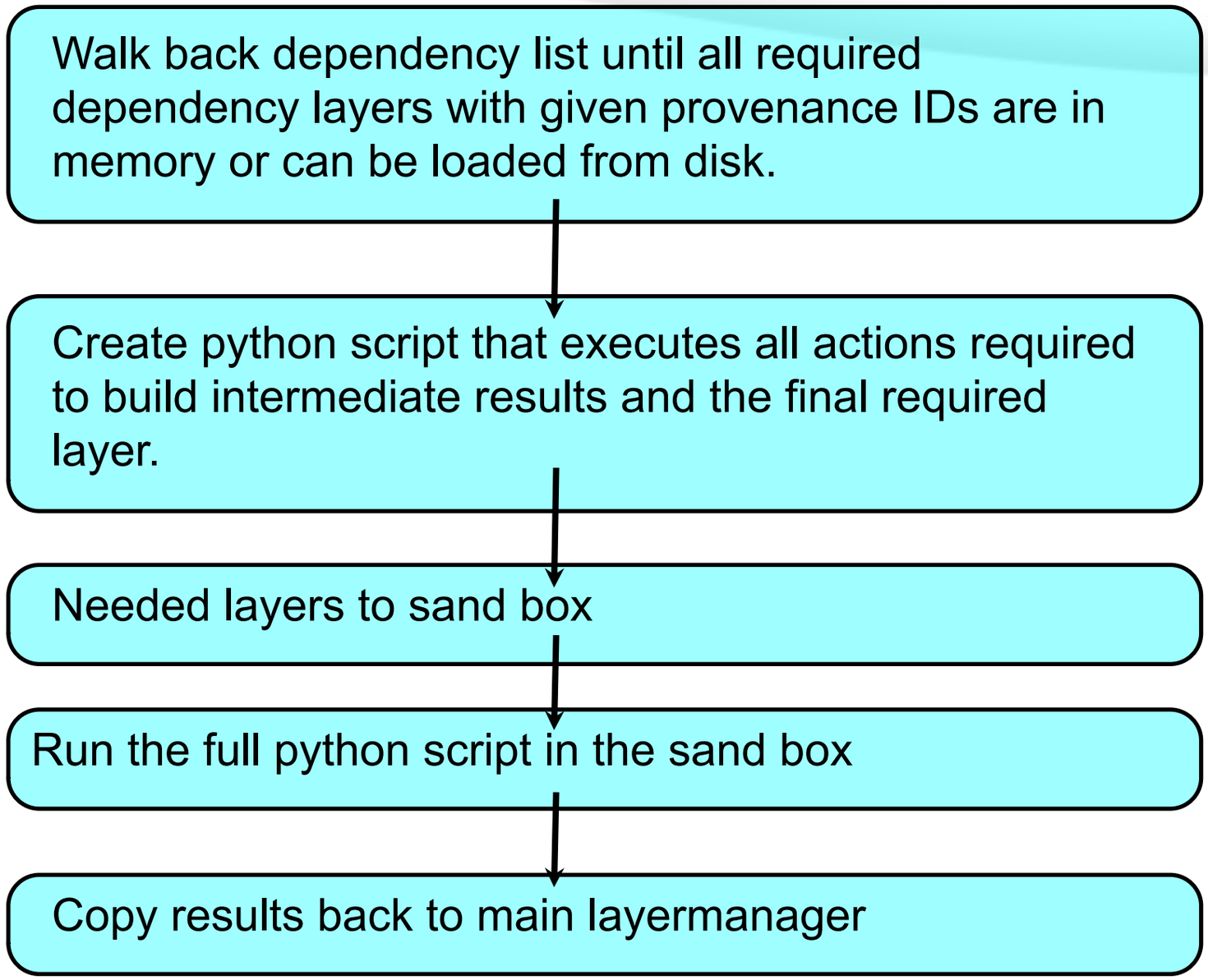
Database records all the layer actions:

* Who did what when and how

* We record a full dependency graph

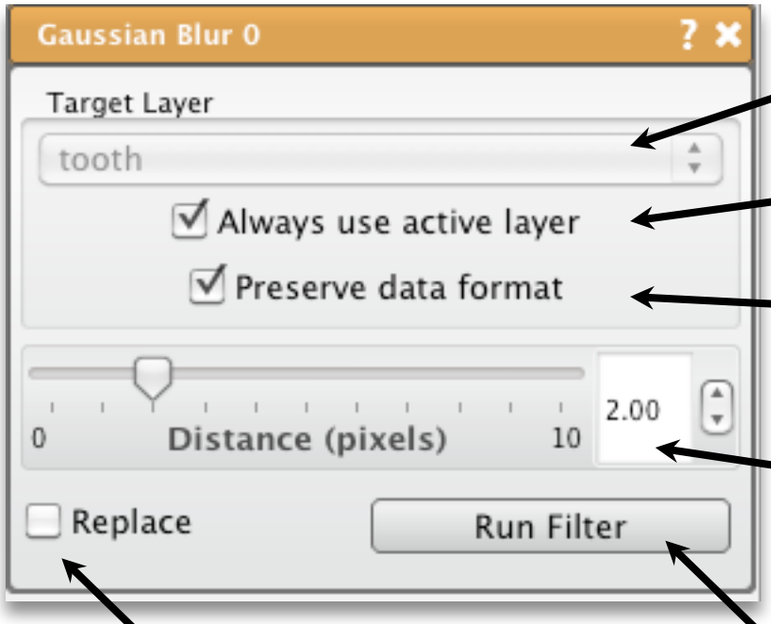
Architecture of Seg3D: Provenance Playback

- Play back is done in sand box to ensure that UI does not need to be updated every time.
- Play back is using python scripts so a future version can use the provenance scripts as a first version for the user to make scripts.



Example 1: GaussianBlurFilter

Filter Example: Overview



Which DataLayer to use

Whether to use the current active layer

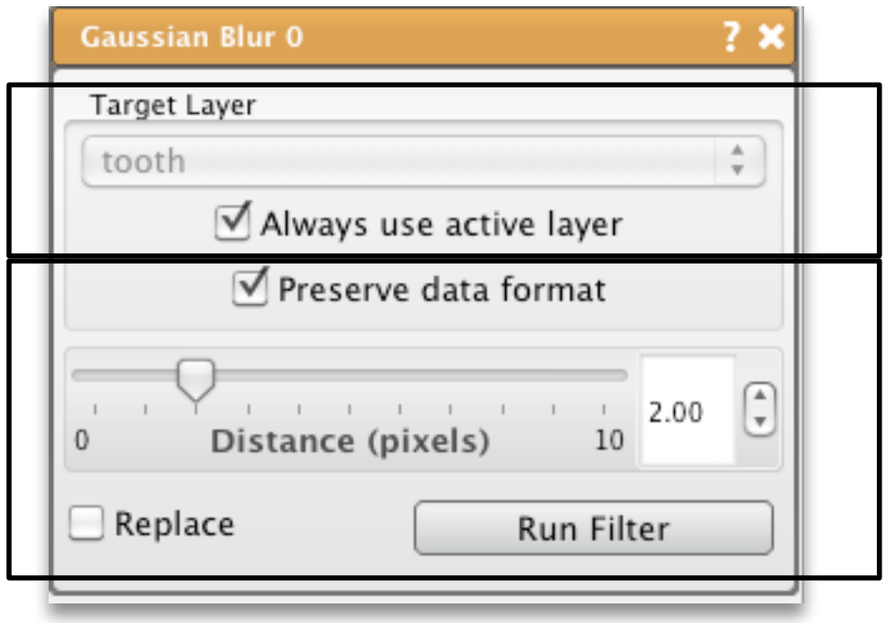
Whether the data format of the layer should be preserved or whether it needs to be converted floating point values

Blurring distance

Button for running the filter

Whether to replace the current layer

Filter Example: Tool Class



Options common to many filters, hence we have a base class that defines these called:
SingleTargetTool

Specific options only needed for this filter, hence these are specified in the derived filter.

Filter Example: SingleTargetTool

We have defined several Tool categories, which defines common behavior among tools. The most common is the single target tool. This class ensures that the execute button and the actively selected layer are managed.

file: Application/Tool/SingleTargetTool.h

```
// Class definition
class SingleTargetTool : public Tool
{

    // -- constructor/destructor --
public:
    SingleTargetTool( int target_volume_type, const std::string& tool_type );

    virtual ~SingleTargetTool();

    // -- state --
public:
    // Layer ID of the target layer
    Core::StateLabeledOptionHandle target_layer_state_;

    // Whether to use the active of one from the list
    Core::StateBoolHandle use_active_layer_state_;
    Core::StateBoolHandle valid_primary_target_state_;

    // Whether a valid layer has been selected
    Core::StateBoolHandle valid_target_state_;

    // Add a state whose input is linked to the target.
    void add_extra_layer_input( Core::StateLabeledOptionHandle input_layer_state,
          Core::VolumeType type, bool required = false, bool dependent = true );

private:
    SingleTargetToolPrivateHandle private_;

};
```

State Variables that describe common features

The name of the layer that needs filtering

Whether to use the active layer

Internal state that keeps track whether a valid was selected.

Whether a valid layer is selected currently. This state decides whether the execute button is activated.

Filter Example: Tool class

file: Application/Tools/DiscreteGaussianFilter.h

```
#include <Application/Tool/SingleTargetTool.h>

namespace Seg3D
{

class DiscreteGaussianFilter : public SingleTargetTool
{
SEG3D_TOOL(
SEG3D_TOOL_NAME( "DiscreteGaussianFilter", "Filter for smoothing data" )
SEG3D_TOOL_MENULABEL( "Gaussian Blur" )
SEG3D_TOOL_MENU( "Data Filters" )
SEG3D_TOOL_SHORTCUT_KEY( "CTRL+ALT+D" )
SEG3D_TOOL_URL( "http://www.sci.utah.edu/SCIRunDocs/index.php/CIBC:Seg3D2:DiscreteGaussianBlur:1" )
SEG3D_TOOL_VERSION( "1" )
)

public:
    DiscreteGaussianFilter( const std::string& toolid );
    virtual ~DiscreteGaussianFilter();

    // -- state --
public:
    // Whether the layer needs to be replaced
    Core::StateBoolHandle replace_state_;

    // Whether the data format needs to be preserved in the filter
    Core::StateBoolHandle preserve_data_format_state_;

    // Blurring distance
    Core::StateRangedDoubleHandle blurring_distance_state_;

    // -- execute --
public:
    // Execute the tool and dispatch the action
    virtual void execute( Core::ActionContextHandle context );

};

}
```

Information block for the ToolFactory. This macro adds functions to the Tool that overload functions in the base class, so the factory can probe properties such as where to add the filter in the menu.

State Variables that record state of the Tool, together with the state of the SingleTargetTool they complete all the information stored in the session file.

Function that executes the filter

Filter Example: Tool class (Implementation)

file: Application/Tools/DiscreteGaussianFilter.cc

```
// Register the tool into the tool factory
SCI_REGISTER_TOOL( Seg3D, DiscreteGaussianFilter )
```

Macro that ensures that a Tool is loaded

```
namespace Seg3D
{

DiscreteGaussianFilter::DiscreteGaussianFilter( const std::string& toolid ) :
    SingleTargetTool( Core::VolumeType::DATA_E, toolid )
{
    // Need to set ranges and default values for all parameters
    this->add_state( "replace", this->replace_state_, false );
    this->add_state( "preserve_data_format", this->preserve_data_format_state_, true );
    this->add_state( "blurring_distance", this->blurring_distance_state_, 2.0, 0.0, 10.0, 0.10 );
}
```

This filter is only working for Data layers.

Linking in the state variables into the StateHandler class and setting up default values as well as identifiers for the session file.

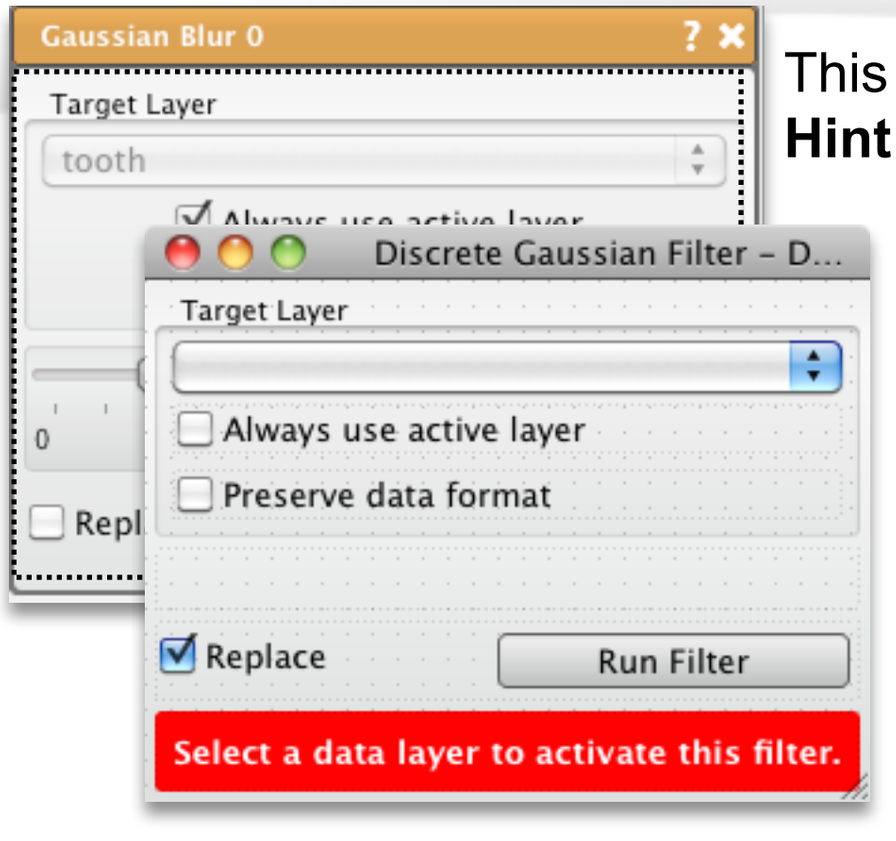
```
DiscreteGaussianFilter::~DiscreteGaussianFilter()
{
    disconnect_all();
}
```

Disconnecting all signals/slots - cannot do this automatically in C++ - a warning will appear in log file, if not included here, but program will work fine most of the time.

```
void DiscreteGaussianFilter::execute( Core::ActionContextHandle context )
{
    // NOTE: Need to lock state engine as this function is run from the interface thread
    Core::StateEngine::lock_type lock( Core::StateEngine::GetMutex() );

    ActionDiscreteGaussianFilter::Dispatch( context,
          this->target_layer_state_->get(),
          this->replace_state_->get(),
          this->preserve_data_format_state_->get(),
          this->blurring_distance_state_->get() );
}

} // end namespace Seg3D
```

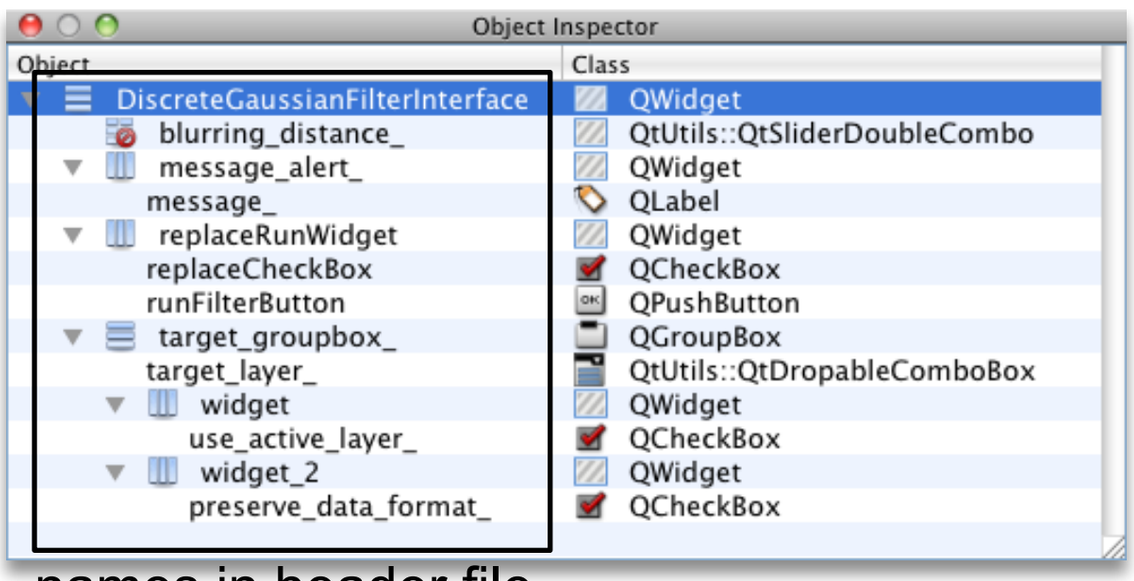
Grab information from the state variables and create a new action

Filter Example: ToolInterface



This part is designed with QtDesigner
Hint: Use an existing ui file as template to make a new filter.

When compiling Seg3D ui files are translated into header files that reside in the bin directory



names in header file

ui_DiscreteGaussianFilterInterface.h

```
class Ui_DiscreteGaussianFilterInterface
{
public:
    QVBoxLayout *verticalLayout;
    QGroupBox *target_groupbox_;
    QVBoxLayout *verticalLayout_4;
    QtUtils::QtDropableComboBox *target_layer_;
    QWidget *widget;
    QHBoxLayout *horizontalLayout_3;
    QCheckBox *use_active_layer_;
    QWidget *widget_2;
    QHBoxLayout *horizontalLayout_4;
    QCheckBox *preserve_data_format_;
    QtUtils::QtSliderDoubleCombo *blurring_distance_;
    QWidget *replaceRunWidget;
    QHBoxLayout *horizontalLayout;
    QCheckBox *replaceCheckBox;
    QPushButton *runFilterButton;
    QWidget *message_alert_;
    QHBoxLayout *horizontalLayout_5;
    QLabel *message_;

    void setupUi(QWidget *DiscreteGaussianFilterInterface)
    {
        ....
    }
}

namespace Ui {
    class DiscreteGaussianFilterInterface: public
Ui_DiscreteGaussianFilterInterface {};
} // namespace Ui
```

Filter Example: ToolInterface header file

file: Interface/ToolInterface/DiscreteGaussianFilterInterface.h

```
namespace Seg3D
{

// Forward declaration
class DiscreteGaussianFilterInterfacePrivate;

class DiscreteGaussianFilterInterface : public ToolWidget
{
Q_OBJECT

// -- Constructor/destructor --
public:
    DiscreteGaussianFilterInterface();
    virtual ~DiscreteGaussianFilterInterface();

// -- create interface --
public:
    // BUILD_WIDGET:
    // This function builds the actual GUI
    virtual bool build_widget( QFrame* frame );

// -- filter internals --
private:
    boost::shared_ptr< DiscreteGaussianFilterInterfacePrivate > private_;

};

} // end namespace Seg3D
```

Base class that defines the frame

Function that is called to build the interface when the tool is opened.

Filter Example: ToolInterface implementation file

file: Interface/ToolInterface/DiscreteGaussianFilterInterface.cc

```
//QtGui includes
#include "ui_DiscreteGaussianFilterInterface.h"

//Application Includes
#include <Application/Tools/DiscreteGaussianFilter.h>

//QTUtils Includes
#include <QtUtils/Bridge/QtBridge.h>

// Interaface includes
#include <Interface/ToolInterface/DiscreteGaussianFilterInterface.h>

SCI_REGISTER_TOOLINTERFACE( Seg3D, DiscreteGaussianFilterInterface );

namespace Seg3D
{

class DiscreteGaussianFilterInterfacePrivate
{
public:
    Ui::DiscreteGaussianFilterInterface ui_;
};

DiscreteGaussianFilterInterface::DiscreteGaussianFilterInterface() :
    private_( new DiscreteGaussianFilterInterfacePrivate )
{
}

DiscreteGaussianFilterInterface::~DiscreteGaussianFilterInterface()
{
}
```

The file generated through QtDesigner

Registering the interface, so Seg3D can find it
Name of the interface needs to be the name of the tool with Interface appended to the name.

The system matches Tools and Interfaces this way

The UI class generated by Qt hidden in a private class

Creating the private class with the UI in it.

Filter Example: ToolInterface implementation file 2

file: Interface/ToolInterface/DiscreteGaussianFilterInterface.cc

```
bool DiscreteGaussianFilterInterface::build_widget( QFrame* frame )
{
    //Step 1 - build the Qt GUI Widget
    this->private_->ui_.setupUi( frame );
    this->private_->ui_.horizontalLayout_3->setAlignment( Qt::AlignHCenter );
    this->private_->ui_.horizontalLayout_4->setAlignment( Qt::AlignHCenter );

    //Step 2 - get a pointer to the tool
    DiscreteGaussianFilter* tool = dynamic_cast< DiscreteGaussianFilter* > ( this->tool().get() );

    //Step 3 - connect the gui to the tool through the QtBridge
    QtUtils::QtBridge::Connect( this->private_->ui_.target_layer_,
          tool->target_layer_state_ );
    QtUtils::QtBridge::Connect( this->private_->ui_.use_active_layer_,
          tool->use_active_layer_state_ );
    QtUtils::QtBridge::Connect( this->private_->ui_.replaceCheckBox,
          tool->replace_state_ );
    QtUtils::QtBridge::Connect( this->private_->ui_.preserve_data_format_,
          tool->preserve_data_format_state_ );

    QtUtils::QtBridge::Connect( this->private_->ui_.blurring_distance_,
          tool->blurring_distance_state_ );

    QtUtils::QtBridge::Enable( this->private_->ui_.runFilterButton,
          tool->valid_target_state_ );

    QtUtils::QtBridge::Show( this->private_->ui_.message_alert_, tool->valid_target_state_, true );

    QtUtils::QtBridge::Enable( this->private_->ui_.target_layer_,
          tool->use_active_layer_state_, true );

    QtUtils::QtBridge::Connect( this->private_->ui_.runFilterButton, boost::bind(
          &Tool::execute, tool, Core::Interface::GetWidgetActionContext() ) );

    this->private_->ui_.blurring_distance_->set_description( "Distance (pixels)" );

    return true;
}
```

Build the widget

Get the Tool that was created with this interface

Connect Widgets in UI to Tool State Variables

Add basic logic to interface to enable button and show the alert message

Call execute on filter when pressing the run button.

Filter Example: Filter Action, class definition

file: Application/Filters/Actions/ActionDiscreteGaussianFilter.h

```
class ActionDiscreteGaussianFilter : public LayerAction
{
```

Filters always derive from LayerAction and not Action

```
CORE_ACTION(
    CORE_ACTION_TYPE( "DiscreteGaussianFilter", "ITK filter that blurs the data." )
    CORE_ACTION_ARGUMENT( "layerid", "The layerid on which this filter needs to be run." )
    CORE_ACTION_OPTIONAL_ARGUMENT( "replace", "true", "Replace the old layer (true), or add an new layer (false)" )
    CORE_ACTION_OPTIONAL_ARGUMENT( "preserve_data_format", "true", "ITK filters run in floating point precision,"
        " this option will convert the result back into the original format." )
    CORE_ACTION_OPTIONAL_ARGUMENT( "blurring_distance", "2.0", "The amount of blurring." )
    CORE_ACTION_OPTIONAL_ARGUMENT( "sandbox", "-1", "The sandbox in which to run the action." )
    CORE_ACTION_ARGUMENT_IS_NONPERSISTENT( "sandbox" )
    CORE_ACTION_CHANGES_PROJECT_DATA()
    CORE_ACTION_IS_UNDOABLE()
)
```

Information block on an Action. This helps converting the action to Python and sets up reasonable defaults

```
    // -- Constructor/Destructor --
public:
    ActionDiscreteGaussianFilter()
    {
        this->add_layer_id( this->target_layer_ );
        this->add_parameter( this->replace_ );
        this->add_parameter( this->preserve_data_format_ );
        this->add_parameter( this->blurring_distance_ );
        this->add_parameter( this->sandbox_ );
    }
```

Link the parameters of this action to the base class. Needs to be done manually, due to C++ limitations.

```
    // -- Functions that describe action --
public:
    virtual bool validate( Core::ActionContextHandle& context );
    virtual bool run( Core::ActionContextHandle& context, Core::ActionResultHandle& result );
```

Main functions that do validation of the action and the other runs the action

```
    // -- Action parameters --
private:

    std::string target_layer_;
    bool replace_;
    bool preserve_data_format_;
    double blurring_distance_;
    SandboxID sandbox_;
```

Action parameters are direct members of the action class, when they linked into the class they will get default values immediately.

```
    // -- Dispatch this action from the interface --
public:
    // DISPATCH:
    // Create and dispatch action that inserts the new layer
    static void Dispatch( Core::ActionContextHandle context, std::string target_layer, bool replace,
        bool preserve_data_format, double blurring_distance );
};
```

This function creates a new action and posts it to the system. It is a convenience function, called by execute in the Tool class.

Filter Example: Filter Action validate

file: Application/Filters/Actions/ActionDiscreteGaussianFilter.cc

```
bool ActionDiscreteGaussianFilter::validate( Core::ActionContextHandle& context )
{
  // Make sure that the sandbox exists
  if ( !LayerManager::CheckSandboxExistence( this->sandbox_, context ) ) return false;

  // Check for layer existence and type information
  if ( ! LayerManager::CheckLayerExistenceAndType( this->target_layer_,
        Core::VolumeType::DATA_E, context, this->sandbox_ ) ) return false;

  // Check for layer availability
  if ( ! LayerManager::CheckLayerAvailability( this->target_layer_,
        this->replace_, context, this->sandbox_ ) ) return false;

  // If the number of iterations is lower than one, we cannot run the filter
  if( this->blurring_distance_ < 0.0 )
  {
        context->report_error( "The blurring distance needs to be larger than zero." );
        return false;
  }

  // Validation successful
  return true;
}
```

SandBoxes separate different python scripts and provenance playback from the currently loaded data

Check whether layer still exists and is of the right type

Check whether no other filter is working on this layer

Check sanity of parameters they can come from Python directly.

Filter Example: Multi threading filters

Application Thread

Filter Thread

Action::validate

Action::run

lock layers and create new layers

create filter algorithm thread

Make temp data structures

Filter the data

Send progress to UI

LayerManager::InsertVolumeIntoLayer

Make changes to the layer itself

Filter Example: Filter Action run

```
bool ActionDiscreteGaussianFilter::run( Core::ActionContextHandle& context,
    Core::ActionResultHandle& result )
{
    // Create algorithm
    boost::shared_ptr<DiscreteGaussianFilterAlgo> algo( new DiscreteGaussianFilterAlgo );

    // Copy the parameters over to the algorithm that runs the filter
    algo->set_sandbox( this->sandbox_ );
    algo->preserve_data_format_ = this->preserve_data_format_;
    algo->blurring_distance_ = this->blurring_distance_;

    // Find the handle to the layer
    if ( !( algo->find_layer( this->target_layer_, algo->src_layer_ ) ) )
      return false;

    if ( this->replace_ )
    {
        // Copy the handles as destination and source will be the same
        algo->dst_layer_ = algo->src_layer_;

        algo->lock_for_processing( algo->dst_layer_ );
    }
    else
    {
        algo->lock_for_use( algo->src_layer_ );

        algo->create_and_lock_data_layer_from_layer( algo->src_layer_, algo->dst_layer_ );
    }

    // Return the id of the destination layer.
    result = Core::ActionResultHandle( new Core::ActionResult( algo->dst_layer_->get_layer_id() ) );

    // Build the undo-redo record
    algo->create_undo_redo_and_provenance_record( context, this->shared_from_this() );

    // Start the filter.
    Core::Runnable::Start( algo );

    return true;
}
```

Create an algorithm that will run the filter

Find the actual Layer class and load that into the algorithm

Lock the layer so no other filter can touch it.

Since we are creating a new layer, we only need a read lock on the old one.

Create a new layer and lock it immediately

Register this filter for undo/redo and in the provenance database

Launch thread that does the filtering

Filter Example: Actual ITK filter part 1

```
class DiscreteGaussianFilterAlgo : public ITKFilter
{

 ....

public:

  SCI_BEGIN_TYPED_ITK_RUN( this->src_layer_->get_data_type() )
  {
        // Define the type of filter that we use.
        typedef itk::DiscreteGaussianImageFilter<
                  TYPED_IMAGE_TYPE, FLOAT_IMAGE_TYPE > filter_type;

    // Retrieve the image as an itk image from the underlying data structure
    // NOTE: This only does wrapping and does not regenerate the data.
    typename Core::ITKImageDataT<VALUE_TYPE>::Handle input_image;
    this->get_itk_image_from_layer<VALUE_TYPE>( this->src_layer_, input_image );

    // Create a new ITK filter instantiation.
    typename filter_type::Pointer filter = filter_type::New();

    // Relay abort and progress information to the layer that is executing the filter.
    this->forward_abort_to_filter( filter, this->dst_layer_ );
    this->observe_itk_progress( filter, this->dst_layer_, 0.0, 0.75 );

    // Setup the filter parameters that we do not want to change.
    filter->SetInput( input_image->get_image() );
    filter->SetUseImageSpacingOff();
    filter->SetVariance( this->blurring_distance_ );
```

Base class adds all functionality to get progress out of ITK and adds converts to itk::Image formats

Macro used to template the itk filter for different types. It sets up the code for each supported data type.

Create new ITK filter

Catch itk signals for progress and abort

Setup itk parameters

Filter Example: Actual ITK filter part 2

```
try
{
        filter->Update();
}
catch ( ... )
{
        if ( this->check_abort() )
        {
                this->report_error( "Filter was aborted." );
                return;
        }
        this->report_error( "ITK filter failed to complete." );
        return;
}
```

Run the itk filter

Some itk filter throw exception when aborted, however behavior is not consistent, hence we record an abort flag before aborting an itk filter.

Other itk filters never throw or do not have a function that checks for aborts, hence we check our flag anyway.

```
if ( this->check_abort() ) return;
```

```
    if ( this->preserve_data_format_ )
    {
            this->convert_and_insert_itk_image_into_layer( this->dst_layer_,
                    filter->GetOutput(), this->src_layer_->get_data_type() );
    }
    else
    {
            this->insert_itk_image_into_layer( this->dst_layer_, filter->GetOutput() );
    }
}
SCI_END_TYPED_ITK_RUN()
```

Only converts data if needed

Integration of state is done on the Application thread, by posting an event to the main thread. Hence the actual integration is done most likely after this thread exits.

Building Filters

- Pick a filter with similar outputs and inputs (helps with building UI)
- Copy and modify the filter to do what you want.

- There are examples for both ITK and Teem filters
- Some filters are implemented directly into C++
- Choose one of the following base classes for the filter:
  - Application/Filters/LayerFilter.h
  - Application/Filters/ITKFilter.h
  - Application/Filters/NrrdFilter.h
- Filter base classes should have most functionality to convert images and to link in observers etc.

Importer Example: VFF Importer

in file: Application/LayerIO/VFFLayerImporter.h

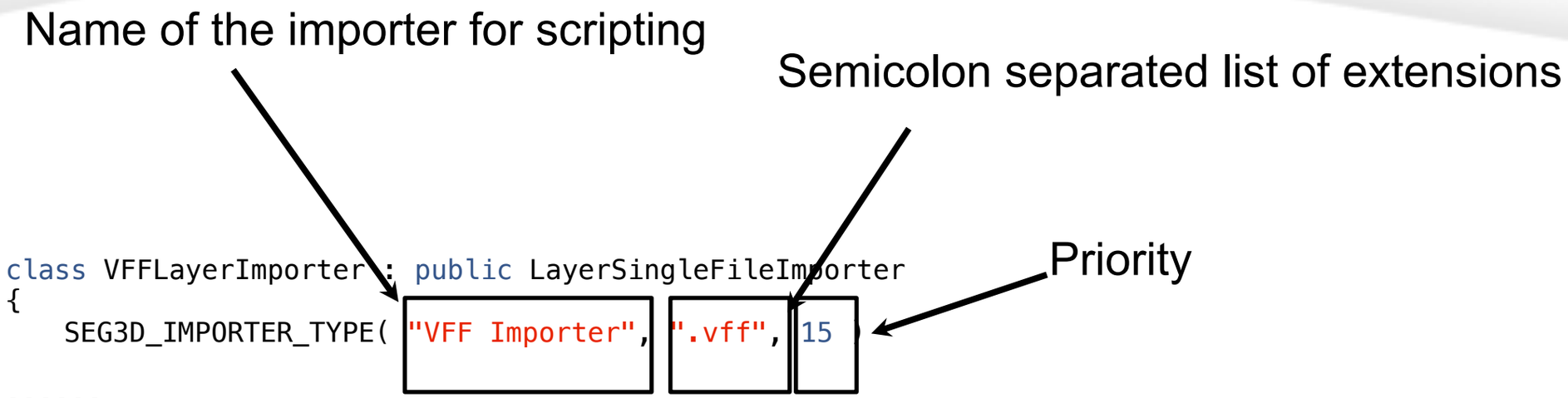
```
class VFFLayerImporter : public LayerSingleFileImporter
{
    SEG3D_IMPORTER_TYPE( "VFF Importer", ".vff", 15 )

    // -- Constructor/Destructor --
public:
    VFFLayerImporter();
    virtual ~VFFLayerImporter();

    // -- Import information from file --
public:
    // GET_FILE_INFO
    // Get the information about the file we are currently importing.
    // NOTE: This function often causes the file to be loaded in its entirety
    // Hence it is best to run this on a separate thread if needed ( from the GUI ).
    virtual bool get_file_info( LayerImporterFileInfoHandle& info );

    // -- Import data from file --
public:
    // GET_FILE_DATA
    // Get the file data from the file/ file series
    // NOTE: The information is generated again, so that hints can be processed
    virtual bool get_file_data( LayerImporterFileDataHandle& data );

    // --internals --
public:
    VFFLayerImporterPrivateHandle private_;
};
```

Single file support and File series support have different base classes.

Stage 1: Determine the type of data in the importer.

Stage 2: Import the data

Importer Example: Importer priority



For each file extension find the importers that import the data. Some file types like DICOMs do not have file extension and thus import everything.

Pick the importer with the highest priority first. Hence the DICOM/ITK importer will be chosen last if there is a better importer available

Importer Example: Importer Logic

Select file or files
Optionally set importer directly

Find importer based on extension

Create Importer and upload filenames to importer

Run the importer information stage on files provided

Stage 1: return a LayerImporterFileInfo object

Show user dialog

Run the importer importer stage

Stage 2: return a LayerImporterFileData object

Generate layer from imported data and copy files to project

Importer Example: VFF Importer, stage 1

```
bool VFFLayerImporter::get_file_info( LayerImporterFileInfoHandle& info )
{
    try
    {
        // Try to read the header
        if ( ! this->private_->read_header() ) return false;

        // Generate an information structure with the information.
        info = LayerImporterFileInfoHandle( new LayerImporterFileInfo );
        info->set_data_type( this->private_->data_type_ );
        info->set_grid_transform( this->private_->grid_transform_ );
        info->set_file_type( "vff" );
        info->set_mask_compatible( true );
    }
    catch ( ... )
    {
        // In case something failed, recover from here and let the user
        // deal with the error.
        this->set_error( "VFF Importer crashed while reading file." );
        return false;
    }

    return true;
}
```

Try to read header

Once read fill in the information object

Importer Example: Spacing and origin from header

```
bool VFFLayerImporterPrivate::read_header()
{
  ...

  // Get the dimensions of the data
  std::vector<size_t> dim;
  Core::ImportFromString( vff_values[ "size" ], dim );

  if ( dim.size() != 3 )
  {
     this->importer_->set_error( "Vff file is not a 3D volume." );
     return false;
  }

  // We check to see if the header contained the origin, if so we use it, otherwise
  // we use default value.
  Core::Point origin;
  if( vff_values.find( "origin" ) != vff_values.end() )
  {
     if (!( Core::ImportFromString( vff_values[ "origin" ], origin ) ) )
     {
          origin = Core::Point( 0.0, 0.0, 0.0 );
     }
  }

  // Similar check for spacing.
  Core::Vector spacing;
  if( vff_values.find( "spacing" ) != vff_values.end() )
  {
     if (!( Core::ImportFromString( vff_values[ "spacing" ], spacing ) ) )
     {
          spacing = Core::Vector( 1.0, 1.0, 1.0 );
     }
  }

  // Generate the grid transform that describes the data
  Core::Transform transform( origin, Core::Vector( spacing.x(), 0.0 , 0.0 ),
     Core::Vector( 0.0, spacing.y(), 0.0 ), Core::Vector( 0.0, 0.0, spacing.z() ) );

  this->grid_transform_ = Core::GridTransform( dim[ 0 ], dim[ 1 ], dim[ 2 ], transform );
  this->grid_transform_.set_originally_node_centered( false );

...
```

Importer Example: VFF Importer, stage 2

```
bool VFFLayerImporter::get_file_data( LayerImporterFileDataHandle& data )
{
    try
    {
        // Read the data from the file
        if ( !this->private_->read_data() ) return false;

        // Create a data structure with handles to the actual data in this file
        data = LayerImporterFileDataHandle( new LayerImporterFileData );
        data->set_data_block( this->private_->data_block_ );
        data->set_grid_transform( this->private_->grid_transform_ );
        data->set_name( this->get_file_tag() );
    }
    catch ( ... )
    {
        // In case something failed, recover from here and let the user
        // deal with the error.
        this->set_error( "VFF Importer crashed when reading file." );
        return false;
    }

    return true;
}
```

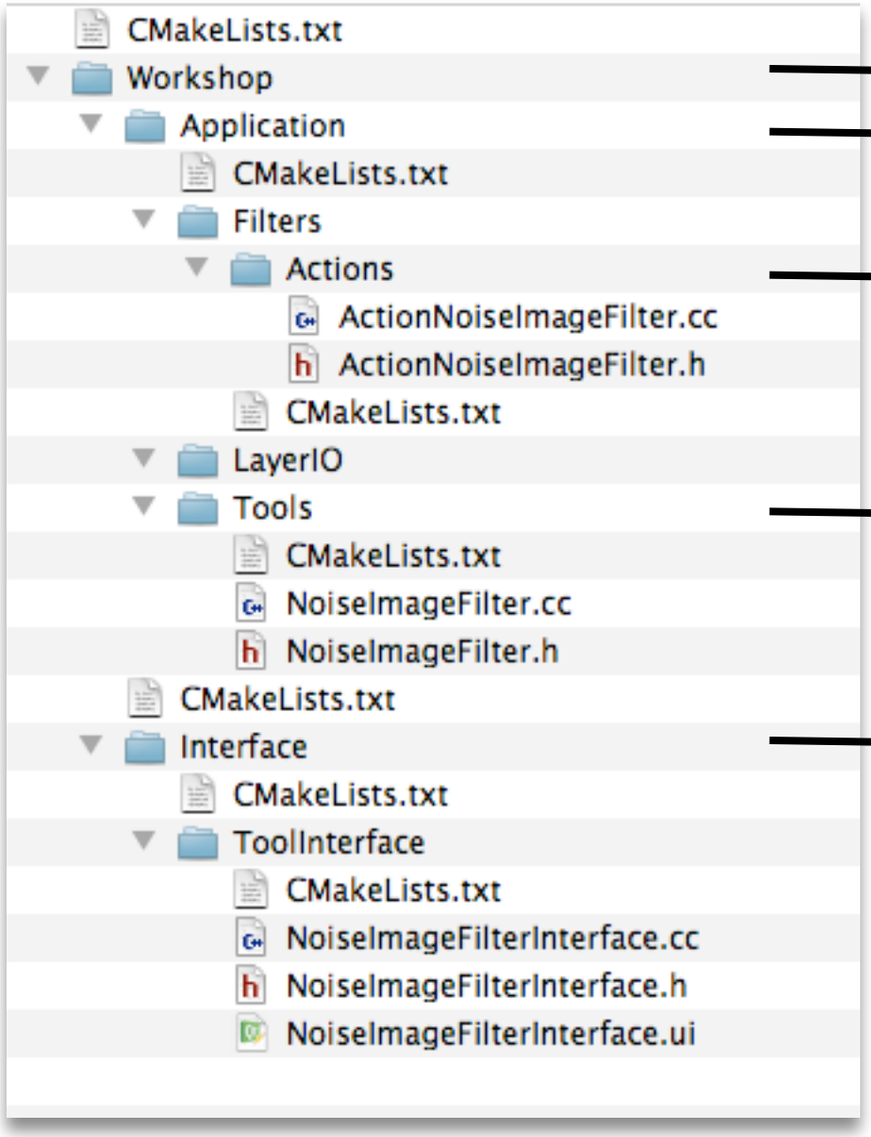
Read the actual data

Once read fill in the data object

Importer Example: Creating data block

```
bool VFFLayerImporterPrivate::read_data()
{
  // Check if we already read the data.
  if ( this->read_data_ ) return true;

  // Ensure that we read the header of this file.
  if ( ! this->read_header() )
  {
     this->importer_->set_error( "Failed to read header of vff file." );
     return false;
  }

  // Generate a new data block
  this->data_block_ = Core::StdDataBlock::New( this->grid_transform_.get_nx(),
     this->grid_transform_.get_ny(), this->grid_transform_.get_nz(), this-
>data_type_ );

  // We need to check if we could allocate the destination datablock
  if ( !this->data_block_ )
  {
     this->importer_->set_error( "Could not allocate enough memory to read vff
file." );
     return false;
  }
 ...
```

Architecture of Seg3D: Plugin Structure



Plugin package

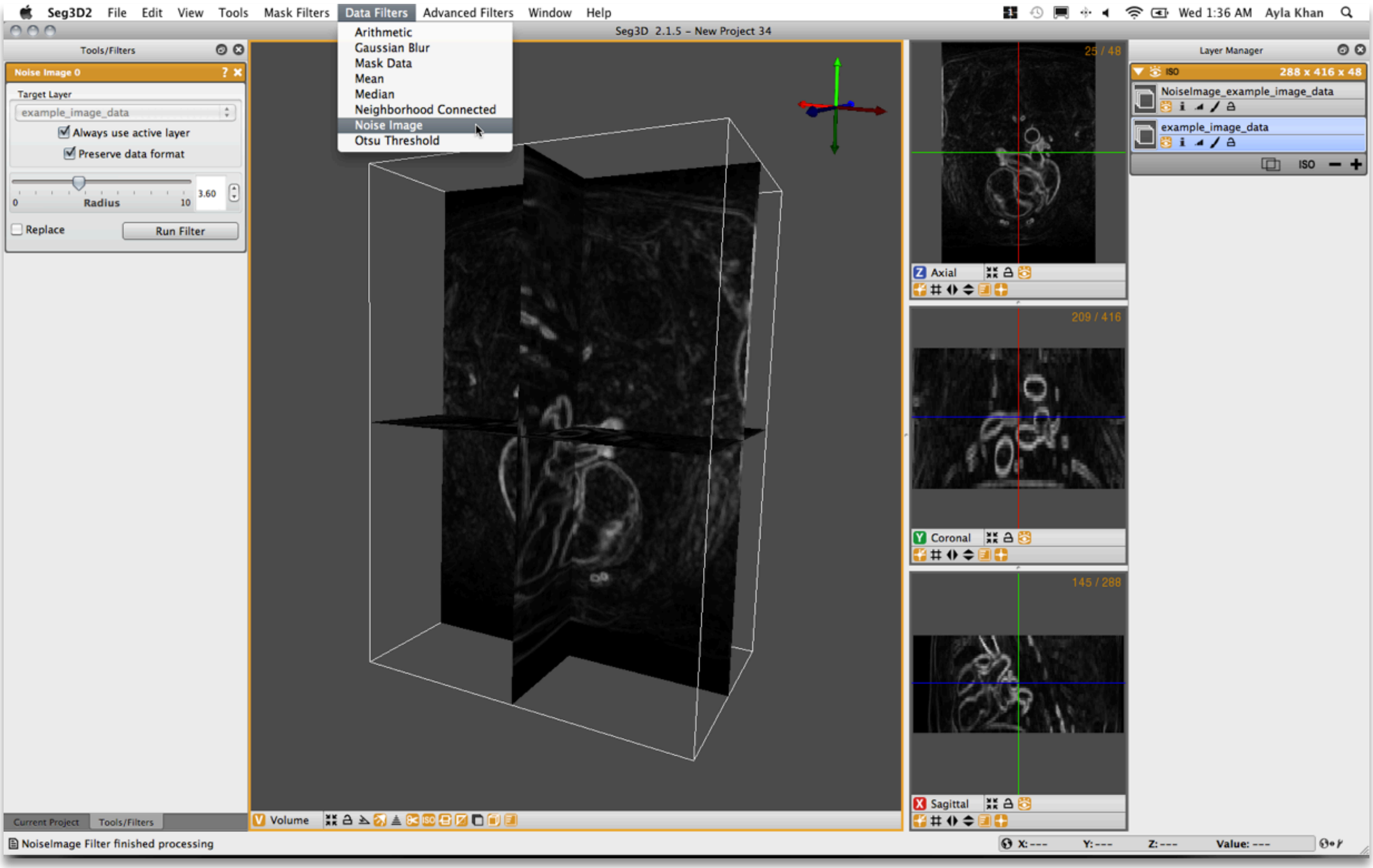
Application files

Actions: filter implementation

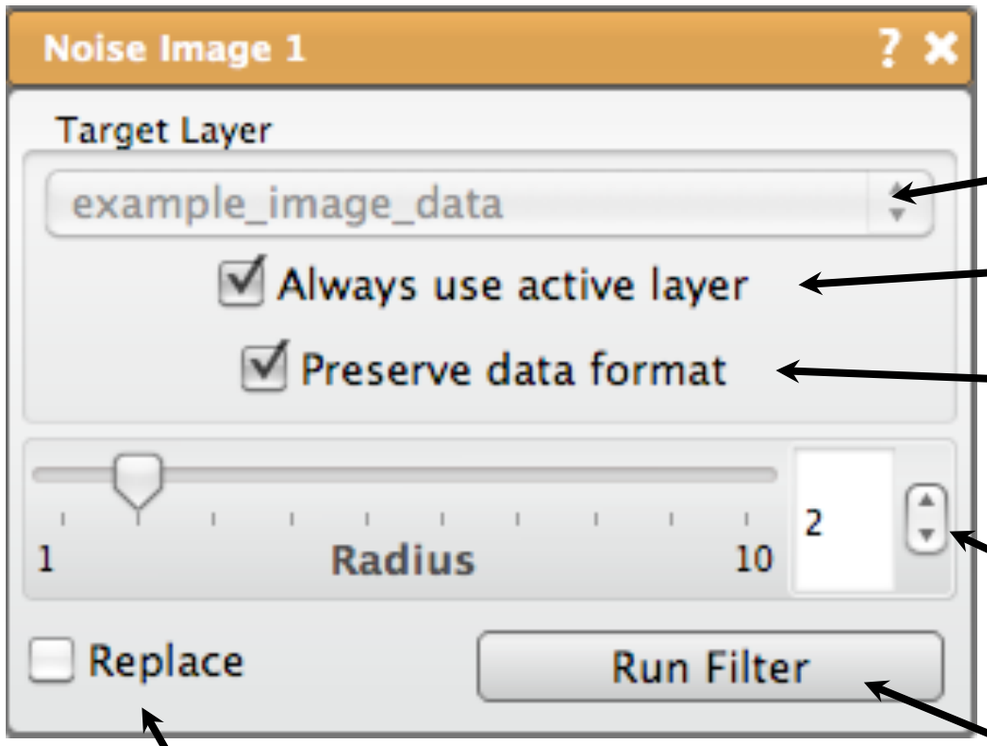
Tools: declare parameters used and link state to current session

Interface (GUI) files: connect widgets to underlying state variables

Plugin Filter in Seg3D



Plugin Filter Overview



Which DataLayer to use

Whether to use the current active layer

Whether the data format of the layer should be preserved or whether it needs to be converted floating point values

Filter radius (size of box filter neighborhood)

Button for running the filter

Whether to replace the current layer

CMake Build: Filter GUI

```
##################################################
# Set sources
##################################################
SET(INTERFACE_TOOLINTERFACE_SRCS
  NoiseImageFilterInterface.cc
)
SET(INTERFACE_TOOLINTERFACE_MOC_SRCS
  NoiseImageFilterInterface.h
)
SET(INTERFACE_TOOLINTERFACE_UI_SRCS
  NoiseImageFilterInterface.ui
)

##################################################
# Generate header out of UI code
##################################################
QT4_WRAP_UI(INTERFACE_TOOLINTERFACE_QT_UI_SRCS ${INTERFACE_TOOLINTERFACE_UI_SRCS})

##################################################
# Wrap QT code to expand all the moc code
##################################################
QT4_WRAP_CPP(INTERFACE_TOOLINTERFACE_QT_MOC_SRCS ${INTERFACE_TOOLINTERFACE_MOC_SRCS})

##################################################
# Ensure that we can find the files generated
# by the moc and ui builder
##################################################

INCLUDE_DIRECTORIES(${CMAKE_CURRENT_BINARY_DIR})

##################################################
# Build the Components library
##################################################
CORE_ADD_LIBRARY(Workshop_Interface_ToolInterface
            ${INTERFACE_TOOLINTERFACE_SRCS}
            ${INTERFACE_TOOLINTERFACE_QT_UI_SRCS}
            ${INTERFACE_TOOLINTERFACE_QT_MOC_SRCS}
            ${INTERFACE_TOOLINTERFACE_NO_MOC_SRCS}
            ${INTERFACE_TOOLINTERFACE_MOC_SRCS})

TARGET_LINK_LIBRARIES(Workshop_Interface_ToolInterface
                      Core_Utils
                      Core_EventHandler
                      Core_Application
                      Core_Interface
                      Core_Action
                      Application_Tool
                      QtUtils_Utils
                      QtUtils_Widgets
                      QtUtils_Bridge
                      Interface_Application
                      ${QT_LIBRARIES}
                      ${SCI_BOOST_LIBRARY})

# Register action classes
REGISTER_LIBRARY_AND_CLASSES(Workshop_Interface_ToolInterface
            ${INTERFACE_TOOLINTERFACE_SRCS})
```

CMake Build: Filter Library

```
################################################
# Set sources
################################################

SET(APPLICATION_FILTERS_ACTIONS_SRCS
  Actions/ActionNoiseImageFilter.cc
)

IF(BUILD_WITH_PYTHON)
  GENERATE_ACTION_PYTHON_WRAPPER(PYTHON_WRAPPER Workshop_Application_Filters ${APPLICATION_FILTERS_ACTIONS_SRCS})
  SET(APPLICATION_FILTERS_SRCS ${APPLICATION_FILTERS_SRCS} ${PYTHON_WRAPPER})
ENDIF(BUILD_WITH_PYTHON)

CORE_ADD_LIBRARY(Workshop_Application_Filters ${APPLICATION_FILTERS_SRCS}
            ${APPLICATION_FILTERS_ACTIONS_SRCS} )

TARGET_LINK_LIBRARIES(Workshop_Application_Filters
                      Core_Utils
                      Core_Action
                      Core_State
                      Core_Parser
                      Application_Layer
                      Application_Project
                      Application_ProjectManager
                      ${SCI_BOOST_LIBRARY})

# Register action classes
REGISTER_LIBRARY_AND_CLASSES(Workshop_Application_Filters
                                        ${APPLICATION_FILTERS_ACTIONS_SRCS})
```

CMake Build: Tools Library

```
##################################################
# Set sources
##################################################

SET(APPLICATION_TOOLS_SRCS
  NoiseImageFilter.h
  NoiseImageFilter.cc
)


IF(BUILD_WITH_PYTHON)
  GENERATE_ACTION_PYTHON_WRAPPER(PYTHON_WRAPPER Application_Tools ${APPLICATION_TOOLS_ACTIONS_SRCS})
  SET(APPLICATION_TOOLS_NOREGISTER_SRCS ${APPLICATION_TOOLS_NOREGISTER_SRCS} ${PYTHON_WRAPPER})
ENDIF(BUILD_WITH_PYTHON)

CORE_ADD_LIBRARY(Workshop_Application_Tools
                 ${APPLICATION_TOOLS_SRCS}
                 ${APPLICATION_TOOLS_NOREGISTER_SRCS}
                 ${APPLICATION_TOOLS_ACTIONS_SRCS}
                 ${APPLICATION_TOOLS_SHADER_SRCS}
                 ${APPLICATION_TOOLS_SHADER_STRING_SRCS})

TARGET_LINK_LIBRARIES(Workshop_Application_Tools
                      Core_Utils
                      Core_Application
                      Core_Interface
                      Core_Action
                      Core_State
                      Core_Geometry
                      Application_Clipboard
                      Application_Filters
                      Application_Tool
                      Application_ToolManager
                      ${SCI_BOOST_LIBRARY}
                      ${SCI_ITK_LIBRARIES})

# Register tool classes
REGISTER_LIBRARY_AND_CLASSES(Workshop_Application_Tools
                      ${APPLICATION_TOOLS_SRCS}
                      ${APPLICATION_TOOLS_ACTIONS_SRCS})
```